# Compute Can't Handle the Truth: Why Communication Tax Prioritizes Memory and Interconnects in Modern AI Infrastructure

Myoungsoo Jung

*Panmnesia, Inc.*
http://panmnesia.com
mj@panmnesia.com

arXiv:2507.07223v2 [cs.DC] 13 Jul 2025

**Abstract**

Modern AI workloads, particularly large-scale language models (LLMs) and retrieval-augmented generation (RAG), impose stringent demands on memory resources, inter-device communication, and flexible resource allocation. Since traditional GPU-centric architectures face scalability bottlenecks, inter-GPU communication overhead often dominates runtime, severely limiting efficiency at large scales.

For better understanding of these challenges, this technical report first introduces fundamental AI concepts in an accessible manner, explaining how contemporary models represent and process complex, high-dimensional data. Specifically, we illustrate how Transformer architectures overcame previous modeling constraints and became foundational technologies underlying modern LLMs. We then analyze representative large-scale AI hardware configurations and data center architectures, detailing how the LLMs are executed within these infrastructures and identifying fundamental sources of scalability challenges in hierarchical deployments.

Based on the observations, we redesign a modular and composable data center architecture leveraging Compute Express Link (CXL), which can address the scalability issues of modern AI data centers. Specifically, the redesigned architecture can independently disaggregate and scale memory, compute, and accelerator resources, dynamically allocating them based on specific workload requirements. This report also explores various CXL topologies and hardware configurations to enable accelerator-centric architectures and facilitate efficient resource disaggregation through modular memory pool designs in data centers. Our empirical evaluations across diverse AI workloads demonstrate that this modular approach can improve scalability, memory efficiency, computational throughput, and operational flexibility.

On the other hand, to accommodate diverse accelerators and hardware scales, we explore and integrate dedicated accelerator-optimized interconnect technologies, collectively referred to as XLink, including Ultra Accelerator Link (UALink), NVIDIA's NVLink, and NVLink Fusion. XLink optimizes latency-sensitive intra-accelerator communication through high-throughput, direct connections, whereas CXL enables scalable, coherent inter-node memory sharing. Motivated by this insight, we introduce a hybrid interconnect approach, CXL-over-XLink, designed to minimize unnecessary long-distance data transfers across scale-out domains, improving overall scalability of scale-up architectures while ensuring memory coherence.

Upon establishing the CXL-over-XLink design, we further propose a hierarchical memory architecture that combines accelerator-local memory and flexible external memory pools to address diverse latency and capacity requirements. In this technical report, we also present optimization strategies for lightweight CXL implementations, high-bandwidth memory (HBM), and silicon photonics to efficiently scale these composable architectures. Finally, we evaluate key performance and scalability parameters critical for modern AI infrastructures.

**Keywords** : CXL, NVLink, NVLink Fusion, UALink, AI Infrastructure, Data Centers, Accelerators, GPUs, Machine Learning, Hardware Architecture.

# Contents

# 1. Introduction

Artificial Intelligence (AI), particularly machine learning (ML), has experienced substantial growth over recent decades through cycles of incremental improvements and major breakthroughs [1–4]. Historically, AI advancements primarily stemmed from increased computational capabilities. However, recent progress relies significantly on data availability and advancements in memory management techniques. Such improvements have enabled contemporary AI systems to effectively emulate complex cognitive functions, achieving human-level or superior performance in tasks like image interpretation, natural language understanding, conversational interactions, and creative content generation [5–8].

Modern AI methodologies rely on transforming data into structured numerical representations, such as vectors and matrices, which enable computational models to learn complex patterns. Specifically, AI models iteratively adjust internal parameters during training using optimization techniques like gradient descent [9–13]. The accuracy and effectiveness of these models depend on their capacity to represent data within high-dimensional numerical spaces [14–18]. As datasets grow in size and complexity, the memory and computational demands of AI models increase significantly, surpassing the practical limitations of traditional CPU-centric computing infrastructures.

To accommodate such extensive data and computational demands, specialized hardware accelerators such as graphics processing units (GPUs[1]) have become widely adopted in industry and academia [19–34]. GPUs provide parallel processing capabilities and integrate substantial internal memory resources (e.g., high-bandwidth memory such as HBM3e [19, 35–38]), enhancing their suitability for contemporary AI workloads.

However, as AI models scale to billions or even trillions of parameters [21, 22, 25, 39–46], individual GPU memory capacities are insufficient by design to meet these demands. For instance, Llama 3 405B [7], with a context window of over a hundred thousand tokens, requires more than a hundred terabytes (TB) of total memory to accommodate embeddings, activations, and optimizer states. This demand exceeds the hundred gigabyte (GB) capacity available in current state-of-the-art GPUs such as NVIDIA's GB200 and GB300 [39, 47–49]. Given the variety of models concurrently deployed, modern AI infrastructures utilize thousands to tens of thousands of GPUs collaboratively, which leads to substantial inter-GPU communication overhead. Industry analyses indicate that such communication accounts for 35%–70% of total training time in large-scale AI deployments, severely limiting efficiency and scalability [29, 50–55]. Given that outcomes from AI research conducted across various disciplines have historically been challenging to directly translate into practical daily applications, it is noteworthy that significant real-world impacts, such as those exemplified by OpenAI's ChatGPT, have emerged prominently only within the past two years. This recent shift highlights the unprecedented data exchanges, information volumes, and memory demands associated with recent large-scale AI workloads compared to previous periods.

Therefore, the central challenge in contemporary AI infrastructures is *no longer purely computational but involves managing massive data transfer volumes, extensive memory resources, and intensive communication demands* characteristic of modern AI workloads. Traditional GPU architectures limit memory scalability due to tightly integrated memory controllers, restricting flexible memory expansion. External memory access via PCIe or storage introduces significant latency, typically ranging from hundreds of nanoseconds ($ns$) to tens of microseconds ($\mu s$), substantially reducing GPU utilization and overall performance [23, 24, 56–58]. To address these limitations, *Compute Express Link* (CXL [59–61]) has emerged as a transformative memory and interconnect technology [23, 56, 62–67]. CXL can fundamentally redefine AI infrastructure by decoupling memory controllers from computational units, enabling independent and dynamic memory management. By externalizing memory controllers and aggregating memory into composable pools accessible by multiple computing nodes, CXL can significantly expand available memory capacity and reduce data communication latency. We believe that this approach can effectively resolve traditional scalability bottlenecks associated with large-scale AI deployments.

To address the challenges and opportunities in redesigning AI infrastructures, it is essential to thoroughly understand the complete system stack, ranging from high-level theoretical AI models and architectures to the low-level hardware configurations deployed in data centers. In this technical report[2], we first provide a clear explanation of fundamental AI principles to comprehensively analyze the scalability challenges of AI. We then describe how recent models represent and process complex, high-dimensional data. We then systematically analyze key factors that have enabled the successful evolution of advanced AI models from the perspectives of

[1] In this technical report, the term 'accelerators' includes GPUs, NPUs, and other specialized processing units. Although GPUs are primarily discussed, all accelerators mentioned can be collectively categorized as coprocessors and treated equivalently as data-processing acceleration hardware.

[2] This technical document, excluding the section pertaining to XLink, is based entirely on the keynote speech delivered by Panmnesia, at the 2024 Summer Conference of the Institute of Semiconductor Engineers.

data management, memory architectures, and optimized communication structures in AI infrastructures. To this end, we examine real-world AI deployments within large-scale data center architectures, focusing on how advanced GPU technologies are utilized and identifying architectural limitations that arise in these deployments.

Subsequently, considering the growing complexity and diverse performance requirements of modern AI workloads, we redesign a modular and composable data center architecture optimized for flexible and dynamic resource management through CXL. True composability requires effective disaggregation of memory, computational units, and accelerator resources. In such composable systems, resources can be independently scaled and precisely adapted to varying workload demands, being able to improve flexibility, scalability, and resource efficiency. Note that CXL technology is central to realizing this composable infrastructure. By enabling coherent memory sharing and scalable interconnect topologies, CXL supports dynamic allocation of memory and computational resources independently of CPU involvement. This architectural advancement substantially reduces latency, enhances memory utilization, and can address a wide range of AI workloads, from general training and inference tasks to specialized applications such as retrieval-augmented generation (RAG) [41, 42, 68–71] and key-value (KV) caching [26, 28, 72–75].

On the other hand, we also propose a strategic integration of complementary interconnect solutions, specifically *accelerator-centric interconnect link* (XLink), an umbrella term encompassing both *Ultra Accelerator Link* (UALink [76–78]) and NVIDIA's *NVLink/NVLink Fusion* [79–83]. XLink technologies optimize latency-sensitive intra-accelerator communication with high throughput and direct connections, but these technologies differ significantly in deployment scope and compatibility: NVLink exclusively targets NVIDIA GPU clusters, while UALink offers an open, Ethernet-based interconnect solution that may exclude NVIDIA GPUs. Meanwhile, both NVLink and UALink typically utilize single-hop Clos topologies, restricting their scalability to rack-level scale-up domains. To bridge this gap and integrate diverse accelerator clusters, we propose a hybrid architecture, *CXL-over-XLink*, in which CXL serves as an inter-cluster fabric. By interconnecting accelerator clusters composed of either NVLink or UALink via CXL fabrics, this architecture enables coherent memory sharing and communication scalability beyond rack-level constraints. This approach can fundamentally reduce unnecessary long-distance communication, including *remote direct memory access* (RDMA)-based data exchanges [84–87], thereby providing scalable architectures tailored for modern AI infrastructures.

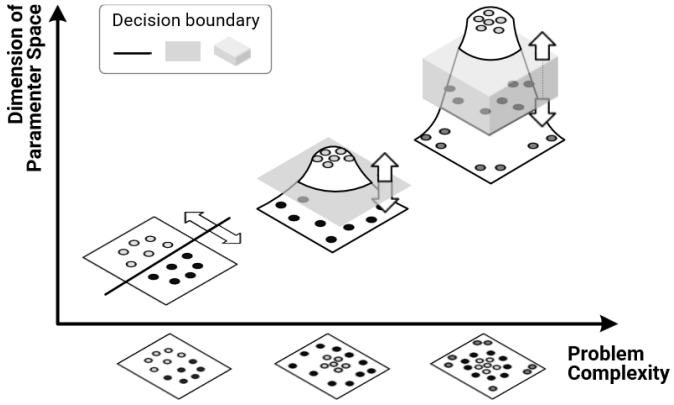
Upon establishing the CXL-over-XLink architecture, we further introduce a hierarchical memory structure to efficiently address diverse latency and capacity demands of contemporary AI workloads. Specifically, we propose integrating accelerator-local memory with lightweight, customized CXL interconnected together with XLink, providing rapid and cache-coherent access to frequently used, latency-sensitive datasets. In parallel, large-scale composable memory pools, connected via lightweight CXL interfaces, handle extensive datasets with relatively relaxed latency constraints. By employing this hybrid memory approach in CXL-over-XLink, our proposed architecture can effectively balance low-latency data access and scalable memory capacity, enhancing overall performance at reduced costs within scale-up domains.

Lastly, to enhance scalability and optimize deployment at larger infrastructure scales, we investigate various hardware-level strategies including HBM, silicon photonics, and cost-efficient, tiered CXL memory implementations. Note that recognizing structural similarities in communication and memory access patterns, we extend our analysis to scientific computing applications, evaluating message passing interface (MPI)-based high-performance computing (HPC) workloads [88–92]. These evaluations moreover illustrate the broad applicability of CXL beyond strictly AI-focused tasks, highlighting its potential to enhance computational efficiency and performance across diverse computational domains.

## 2. From RNNs to Transformers: Evolution in Sequence Modeling

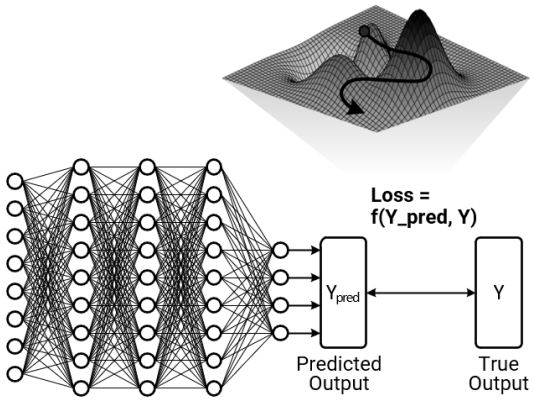
The purpose of this section is to explain why AI, previously confined largely to research domains, has recently transitioned into practical everyday applications. First of all, we introduce fundamental AI concepts in an accessible manner, highlighting how contemporary models represent and process complex, high-dimensional data, and describe how advancements in hardware acceleration have enabled this transition. To achieve this, we discuss the progression of sequence modeling techniques, beginning with early *Sequence-to-Sequence* (Seq2Seq [93–95]) frameworks, advancing through attention mechanisms, and progressing to *Transformer* architectures [3, 6, 96], ultimately leading to modern *Large Language Models* (LLMs [46, 97–101]).

Specifically, in this section, we examine the practical importance and characteristics of time-series data, introducing foundational concepts underlying the Seq2Seq framework. We then address the strengths and limitations of early Seq2Seq implementations based on *Recurrent Neural Network* (RNN [102–105]), describing how attention mechanisms resolved key shortcomings. Next, we explain how the Transformer architecture leveraged




(a) 2D space. (b) 3D space. (c) 4D space.

Figure 1: Setting the decision boundary.




(a) Many parameters. (b) Gradient descent.

Figure 2: Minimizing the loss function.

advances in "parallel computing" and "hardware acceleration" to overcome scalability limitations of earlier approaches, enhancing AI's applicability to real-world scenarios. Lastly, we outline how Transformers evolved into LLMs and emphasize their implications for hardware infrastructure and scalability, both critical factors in their widespread practical adoption.

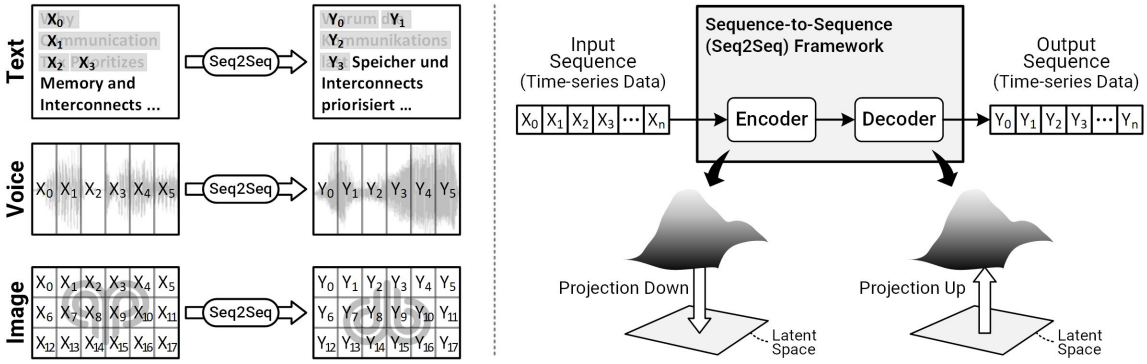
## 2.1. Understanding Time-Series Data and the Sequence-to-Sequence Framework

Recent advances in AI technologies have achieved human-level accuracy across diverse real-world problems, attracting broad attention and driving innovation in both industry and academia. Before discussing sequence models, it is useful to briefly outline the foundational methods of AI model training and inference, particularly regarding the handling of time-series data.

**Brief overview of AI training and inference methods.** To solve complex real-world problems, AI typically transforms input data into mathematical representations, such as numbers, vectors, or matrices mapped onto coordinate systems of specific dimensions. As depicted in Figure 1, these transformed data representations allow the establishment of decision boundaries, which categorize input data into distinct groups based on identifiable patterns. The *training* process involves the AI model navigating the parameter space to optimize these decision boundaries, enhancing their clarity and effectiveness. *Inference*, conversely, refers to determining the category of new inputs based on established decision boundaries, with accuracy measured by the similarity between the model's prediction and the actual outcome.

Determining a decision boundary during AI model training involves measuring how the predicted results differ from true values, through a metric known as a *loss function*, which will be explained in detail shortly. The optimal decision boundary minimizes this loss, resulting in high inference accuracy and effective resolution of real-world problems. However, as real-world problems grow more complex, clearly separating data groups in low-dimensional spaces becomes challenging. For instance, as illustrated in Figure 1a, a two-dimensional plane (2D) may distinguish two groups easily using a simple linear boundary in ideal cases, but such simple scenarios are uncommon in practice. To address this issue, AI models expand their parameter spaces into higher dimensions, such as 3D and 4D (Figures 1b and 1c), allowing clearer and more precise representation of complex data. Although the illustrated examples focus on a limited number of dimensions, modern large-scale AI models use parameter spaces consisting of billions to trillions of dimensions [7, 25, 106–109]. These extensive parameter spaces greatly enhance the models' ability to solve previously intractable problems but also significantly increase memory and communication requirements.

Defining effective decision boundaries also requires knowledge of the characteristics and roles of loss functions. Commonly used loss functions include mean squared error loss [110–113] and cross-entropy loss [114–118], as mentioned earlier. As illustrated in Figure 2, the goal of AI model training is to configure model parameters such that the loss function achieves its minimum value. However, even when the dimensions of data representation could be optimally determined, the sheer number of parameters in AI models often makes it impractical to analytically find an exact minimum. Therefore, approximate algorithms such as *gradient descent* [110, 119–121] are widely employed. Gradient descent starts from arbitrary initial parameter values and iteratively updates




(a) Time-series data sequences.

(b) Fundamental concept of Seq2Seq model.

Figure 3: Sequence-to-sequence (Seq2Seq) framework.

these parameters based on gradient information, moving toward lower loss values. The algorithm terminates upon reaching a point where the gradient equals zero, indicating the minimum of the loss function.

Despite its effectiveness, gradient descent exhibits certain limitations. The step size, or the magnitude of parameter adjustments, is important: excessively large steps can overshoot minima, while excessively small steps prolong convergence. Moreover, initiating the search from arbitrary points may lead to convergence at local minima rather than the global minimum. To address these limitations, various optimization techniques, such as stochastic gradient descent (SGD [110, 122–128]) or adaptive moment estimation (Adam [129–133]), have been developed, dynamically adjusting step sizes and search directions. Note that these optimizations, while varied, ultimately share the common goal of adjusting the model parameters so that the loss function reaches its minimum, which aligns with the aforementioned overall objective of AI training.

**Early implementations and limitations: Recurrent neural networks.** In practical AI applications, input data encompass diverse modalities such as images, audio, video, and text. Despite these variations, many real-world datasets possess temporal or sequential characteristics, which allow them to be generalized and analyzed as time-series data. As shown in Figure 3a, transforming different types of input data into structured numerical sequences enables AI models to capture temporal dependencies essential for accurate predictions and sophisticated decision-making. Sequence modeling techniques have emerged specifically to handle these structured sequences.

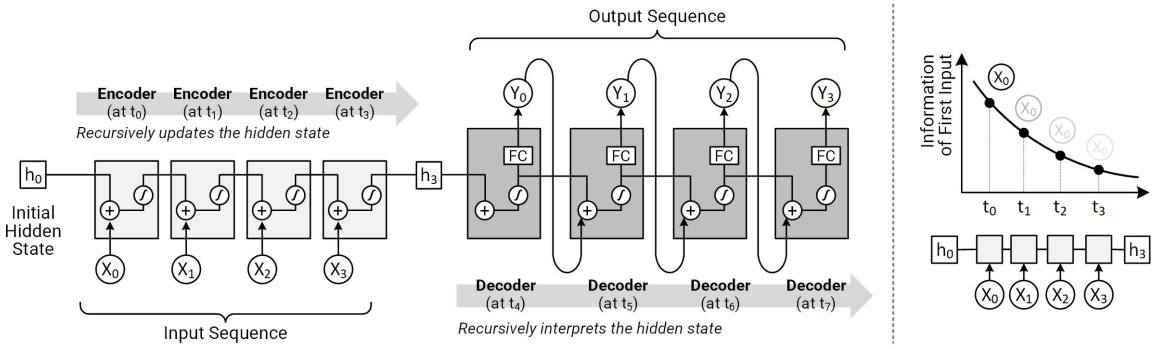
One influential model designed to address this requirement is the Sequence-to-Sequence (Seq2Seq) framework. Introduced in 2014 [93], Seq2Seq remains fundamentally significant today as the foundation for many advanced sequence modeling approaches. This section briefly describes its core structure. Specifically, the concepts of Encoding, Ordering, and Decoding – essential for understanding Seq2Seq – are schematically illustrated in Figure 3b and discussed below:

1. **Encoding:** The encoder ("projecting down") compresses input sequences into concise numerical representations, known as latent *vectors*. These vectors can capture essential context and temporal relationships within data, facilitating efficient sequence analysis.

2. **Ordering:** Maintaining sequential order within internal processing ("sequential processing") ensures each step incorporates context from preceding elements. This ordering preserves coherence and accuracy throughout the generated outputs.

3. **Decoding:** The decoder ("projecting up") reconstructs these internal representations into understandable output sequences. It leverages context-rich latent vectors to generate coherent outputs, such as translations, summaries, or predictive forecasts.

The encoder-decoder framework in Seq2Seq models handles sequences of varying lengths and complexities. By compressing the entire input sequence into concise internal representations, the encoder ensures that contextual and temporal information is preserved. The decoder then leverages these representations to generate accurate and coherent output sequences, bridging the gap between complex data patterns and human interpretability.




(a) Encoding process. (b) Decoding process. (c) Vanishing gradient.

Figure 4: RNN-based Seq2Seq architecture and limitation.

Thanks to this structured methodology, Seq2Seq models have become effective tools for various practical tasks, including language translation, summarization, speech recognition, anomaly detection, and predictive analytics.

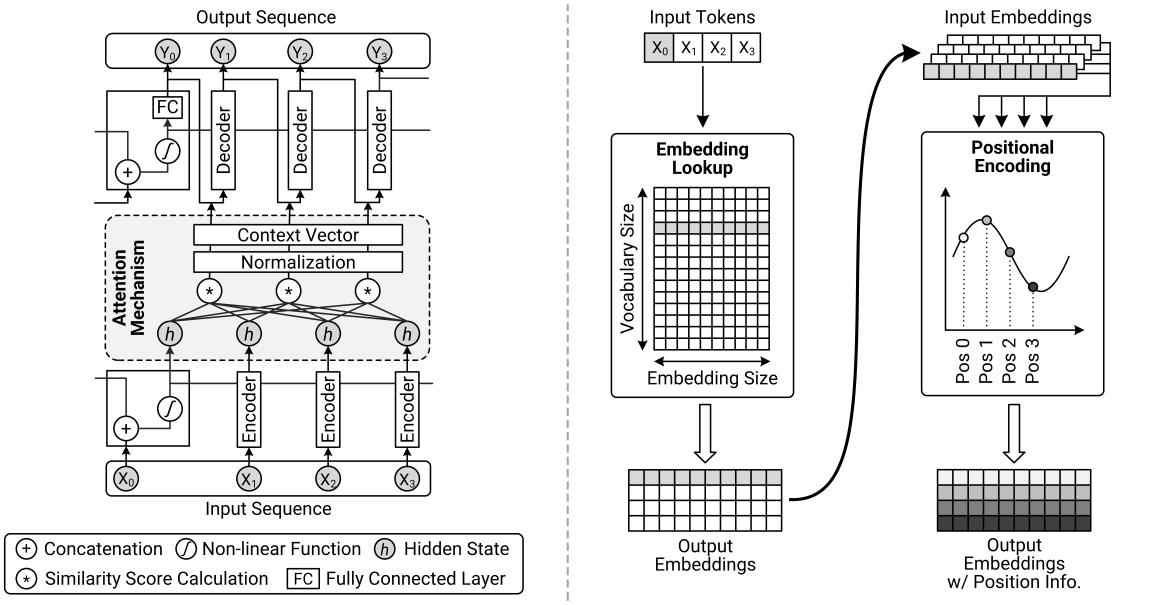
Initial implementations of Seq2Seq models relied on RNNs, which naturally handle sequential data. As shown in Figure 4a, in RNN-based Seq2Seq architectures, the encoder's primary role is to read and understand the input sequence as explained previously. It achieves this by processing each data point sequentially and maintaining an internal memory, known as the *hidden state*, which summarizes the essential context of previously seen data points. At each step, the encoder combines the current data point with the previous hidden state through mathematical operations involving non-linear functions (e.g., tanh and ReLU [134, 135]). These non-linear functions enable the network to capture complex and non-linear relationships within the data. Without such non-linear transformations, the network's ability to model intricate patterns and temporal dependencies in sequential data would be limited [136–138]. This sequential updating allows the encoder to preserve essential context and information from all preceding data points, gradually summarizing the entire input sequence into a condensed, meaningful internal representation.

As shown in Figure 4b, once the encoder completes this task of compressing the sequence, the decoder reconstructs the summarized representation into a comprehensible and structured output sequence. The decoder functions by generating output data points in a sequential manner, using not only the condensed hidden state representation but also its previously generated outputs as inputs for each subsequent step. To perform this reconstruction, the decoder employs *fully connected* (FC) layers [3, 139]. These FC layers are specialized neural network layers designed to map abstract and compressed internal representations back into understandable, real-world outputs. Each layer transforms the internal information into a clearer and more concrete form, allowing the decoder to produce accurate and coherent sequences such as translations, summaries, or predictions.

Despite their initial success, RNN-based Seq2Seq models unfortunately faced serious limitations, which are illustrated by Figure 4c. A prominent challenge is the "vanishing gradient problem [140, 141]," where information from early sequence elements gradually diminishes during training, degrading the model's ability to capture long-range dependencies. Simply put, this is analogous to the human phenomenon of forgetting older memories, or forgetfulness. In addition, *due to their intrinsic sequential nature, RNNs cannot leverage parallel computing technologies*, limiting their scalability and computational efficiency. These constraints motivated the development of more advanced architectures designed to overcome the shortcomings of RNNs by enhancing long-range context retention and enabling parallel computation.

## 2.2. A Paradigm Shift in Sequence Modeling

**Integration of attention mechanisms.** To overcome the limitations of conventional RNN architectures, especially the vanishing gradient problem and poor scalability, researchers introduced *attention* mechanisms [3, 142, 143], marking a significant evolution in sequence modeling. The fundamental idea behind attention is to allow the model to focus on the most relevant parts of the input sequence when generating each element of the output. Instead of compressing the entire input sequence into a single fixed-size hidden state (as done in traditional RNNs), attention mechanisms maintain access to all hidden states produced by the encoder and selectively assign weights to them based on their relevance to the current decoding step.




(a) Attention mechanism over RNNs. (b) Embedding layer. (c) Positional encoding.

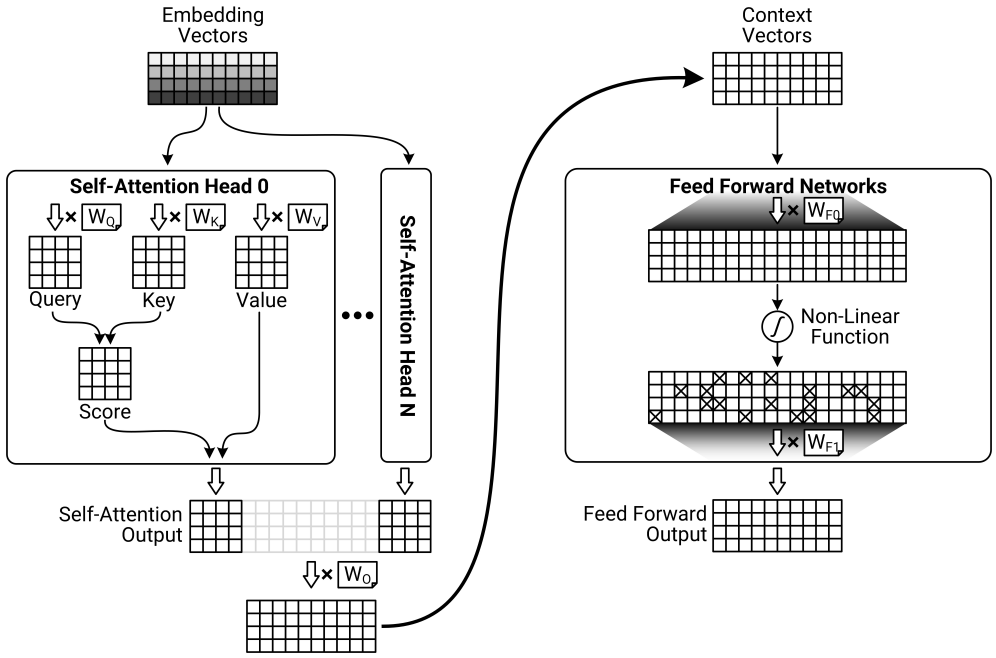
Figure 5: Attention-based RNNs and Transformer architecture.

As shown in Figure 5a, at each decoding time step, the attention mechanism computes a similarity score between the decoder's current hidden state and each encoder hidden state. These scores are then normalized to produce attention weights, typically using a softmax function [144–146]. The decoder uses these weights to compute a weighted sum of the encoder hidden states, resulting in a context vector that captures the most relevant information from the input sequence. This vector is then used, along with the decoder's previous outputs, to generate the next token in the output sequence. Here, a *token* refers to basic data units such as words, subwords, or characters.

This process provides two main advantages. First, attention mechanisms allow models to retain and access information from "any part of the input sequence", regardless of its length. By eliminating the need to compress all information into a fixed-size vector, this approach addresses the vanishing gradient problem and provides models with robust, content-based memory, enabling efficient referencing of relevant sequence elements during output generation [3, 142, 143, 147]. Second, attention enhances model interpretability. Attention weights indicate which parts of the input the model focuses on when generating outputs, offering intuitive insights into the reasoning and decision-making processes of complex models [148–150].

Unfortunately, despite these improvements, models that used attention mechanisms within RNNs still inherited the sequential nature of computation from their underlying architecture (i.e., ordering). Each time step had to be computed in order, which limited the ability to leverage modern parallel processing hardware. This motivated the development of fully attention-based architectures, culminating in the Transformer model, which reshaped the landscape of sequence modeling.

**From recurrence to parallelism: Transformer revolution.** A major breakthrough in sequence modeling occurred in 2017 with the introduction of the Transformer architecture [3], marking a fundamental departure from the recurrent computational structure intrinsic to RNNs and conventional Seq2Seq models. Transformers entirely abandoned recurrence, adopting instead a fully attention-based design. In particular, the Transformer utilizes *self-attention* [151–155], which computes interactions among all positions within a sequence simultaneously. Unlike traditional attention mechanisms embedded within RNNs, self-attention independently calculates attention scores across all sequence positions in parallel, without relying on previously computed states or sequential dependencies between positions. This independence arises because each position's attention calculation is based directly on fixed input representations, known as *embeddings*. By eliminating sequential processing constraints, Transformers exploit modern parallel processing hardware, boosting computational efficiency and scalability [43, 156, 157].




(a) Multi-head self-attention mechanism.

(b) Feed-Forward networks (FFNs).

Figure 6: Transformer layer operations: self-attention and FFNs.

This parallel processing capability is enabled by the Transformer's embedding layers, which project discrete tokens into high-dimensional continuous vector spaces. As shown in Figure 5b, these embeddings serve as input representations for the model, enabling simultaneous processing of all tokens in a sequence. Unlike RNN-based Seq2Seq models, which must process inputs sequentially due to their recurrent structure, Transformers operate directly on embeddings in parallel. Each token's embedding distinctly captures semantic meaning, allowing the self-attention mechanism to relate tokens across the entire sequence in parallel; the details of the self-attention will be explained, shortly. This simultaneous token-level interaction removes the step-by-step dependency inherent in recurrent models, enabling full parallelization of computations across all tokens.

However, the removal of explicit sequential structure introduces a new challenge: Transformers lack a sense of order in the input sequence, also referred to as "ordering". In RNNs, the order is preserved due to their sequential processing nature. To compensate for this, the Transformer introduces *positional encoding*, which is a mechanism that injects information about the relative or absolute position of tokens directly into their embeddings (cf. Figure 5c). These encodings are added to the input embeddings before they are processed by the self-attention layers, ensuring that the model can distinguish between tokens based on their positions in the sequence. Positional encoding can be implemented using fixed sinusoidal functions or learned embeddings [158–160], both of which allow the model to capture sequence order while preserving the ability to compute in parallel.

This combination of embedding-based input representation and positional encoding enables Transformers to model long-range dependencies while fully leveraging parallel computation. As a result, the Transformer architecture achieves superior performance and scalability compared to RNN-based models. It has become the foundation for virtually all modern large-scale language models and sequence modeling tasks.

**Self-attention and mixture of experts: Enhancing sequence understanding.** The self-attention mechanism is a cornerstone of the Transformer architecture, enabling the model to simultaneously relate different parts within a sequence. Unlike traditional attention mechanisms, which typically align separate input-output sequences, self-attention directly computes interactions among tokens within a single sequence. As depicted in Figure 6a, this intra-sequence interaction is facilitated by three distinct vectors, *query*, *key*, and *value*, each derived from individual token embeddings through separate linear transformations using learnable parameter matrices ($W^Q$, $W^K$, and $W^V$). Specifically, query vectors (Q) represent the information that each token seeks,

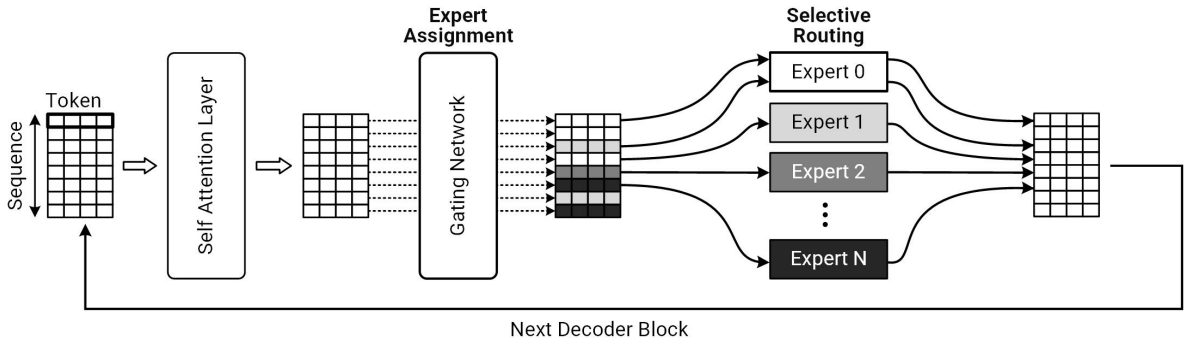



Figure 7: Mixture of experts (MoE) architecture.

key vectors (K) denote information each token provides, and value vectors (V) encapsulate the actual content shared by tokens. The self-attention process involves comparing each query vector with all key vectors using scaled dot-product operations, producing attention scores reflecting token similarities or relevancies. These scores are scaled by the square root of the key dimension, ensuring numerical stability during training, and subsequently normalized via a softmax function to yield attention weights. The context-aware representation for each token is then computed as a weighted sum of all value vectors, with weights determined by these attention scores.

Transformer architectures further enhance representational capability through *multi-head attention* [3, 161, 162] (or grouped query attention [163–165]), where several parallel self-attention computations (*heads*) operate concurrently. Each attention head employs independent parameter matrices to generate distinct query, key, and value vectors, enabling the model to simultaneously capture diverse relationships and nuanced token interactions. This multi-head design broadens the model's ability to capture complex and long-range dependencies, thus enhancing overall model accuracy and contextual understanding without sequential processing constraints.

In addition, as shown in Figure 6b, Transformers incorporate specialized *Feed-Forward Networks* (FFNs) within their architecture, distinct from the FC layers typically mentioned in classical neural networks. While self-attention can capture the contextual relationships among tokens, each token representation requires further refinement. To this end, FFNs apply separate, position-wise nonlinear transformations independently to each token embedding, complementing the self-attention mechanism. Specifically, each FFN comprises two linear transformations separated by a non-linear activation function (e.g., ReLU or GELU [135, 166]). The first linear transformation projects input embeddings into a "higher" dimensional representation space, enabling the model to capture complex, non-linear data patterns. The second linear layer subsequently projects these refined representations back to the "original" dimensional space. Consequently, FFNs enhance the context-aware embeddings produced by self-attention. As these FFN layers operate independently at each token position, they preserve the inherent parallel processing advantages characteristic of the Transformer architecture.

Even though FFNs refine token-level representations produced by self-attention, further improvements in handling diverse and complex data require increased model capacity and computational efficiency. To address this, Transformers incorporate advanced structures such as the *Mixture of Experts* (MoE) architecture [40, 167, 168]. Figure 7 illustrates an overview of MoE, designed to enhance model capacity and efficiency. An MoE model consists of multiple specialized neural networks, called *experts*, each trained to address distinct data types or patterns. During inference, a gating network dynamically routes input tokens to selected experts according to learned patterns, activating only the necessary experts to optimize computational resources. This selective expert activation not only enhances computational efficiency but also improves model accuracy by enabling experts to specialize in particular subtasks. As a result, MoE-based Transformers achieve high scalability, robustness, and generalization, making them effective for large-scale and diverse datasets [40, 168–171].

Note that MoE enables high computational efficiency through parallel processing by multiple independent experts, but aggregating the intermediate outputs of these experts necessitates significant inter-expert communication. Moreover, because the aggregated MoE outputs serve as inputs for subsequent layers such as self-attention, a sequential dependency exists; the self-attention can only begin processing once MoE aggregation is complete. As a result, efficient *AI infrastructure design must include high-speed interconnects to facilitate rapid communication* and support this essential sequential processing order.

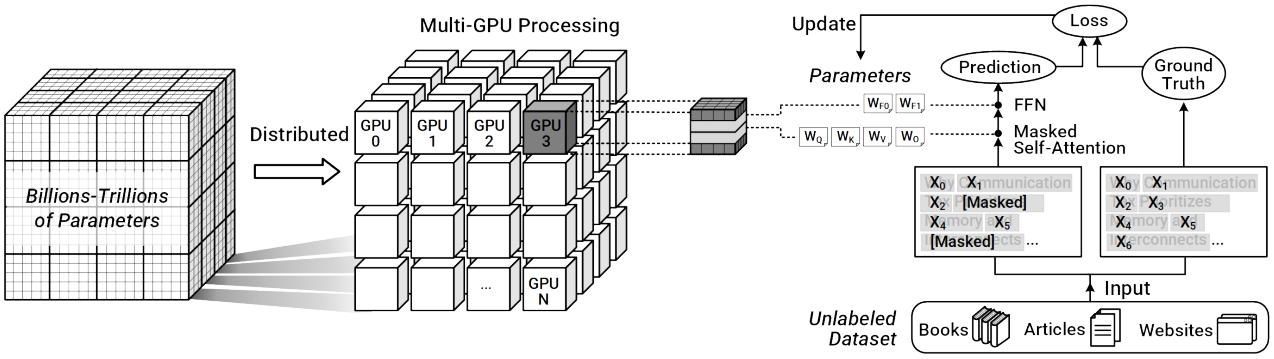



Figure 8: LLM pre-training through parallel processing.

## 2.3. From Transformers to Large Language Models

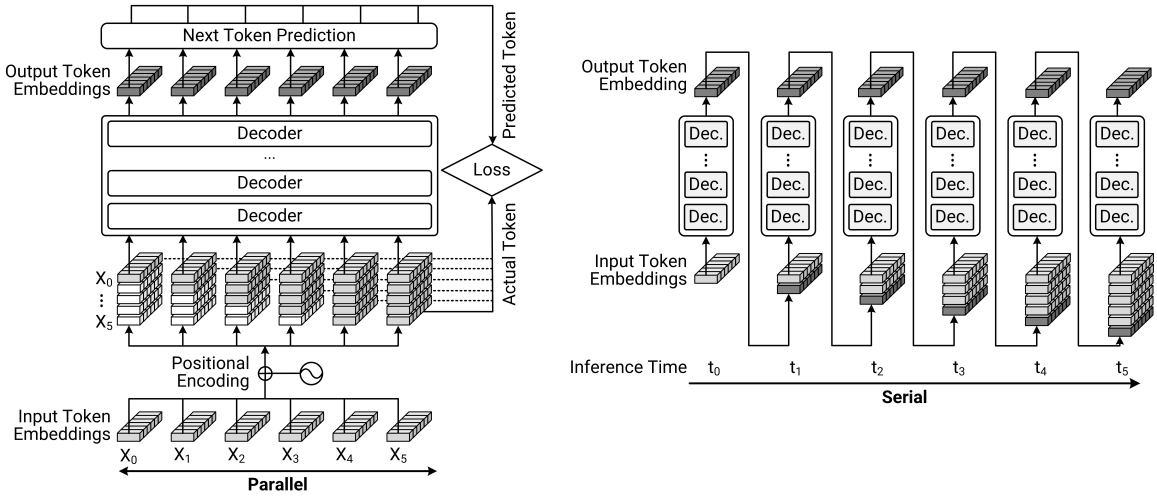
The introduction of Transformers marked a pivotal shift in sequence modeling by replacing the recurrent computations in RNNs with parallelizable self-attention mechanism. This innovation enabled models to concurrently analyze relationships among all tokens in a sequence, accelerating computation and supporting the development of deeper, more complex neural architectures. This transition also served as a critical turning point for the widespread adoption of hardware accelerators such as GPUs, as the underlying computation patterns aligned well with massively parallel hardware.

**Massive-scale training and parameter optimization.** Building upon the scalability of Transformers, researchers developed LLMs, extending model capabilities by increasing their parameter counts and training them extensively on massive and diverse datasets. Modern LLMs, exemplified by models such as GPT-4 Turbo [97, 172] and Google's Gemini [173, 174], can contain billions or even trillions of parameters. These parameters, adjustable internal settings of neural network layers, are optimized during a comprehensive pre-training phase involving vast textual datasets, including books, scholarly articles, websites, and social media content [6, 175, 176]. This pre-training commonly employs self-supervised learning [177, 178], an approach that enables models to derive meaningful representations from unlabeled data in an automatic manner. Typically, this involves predicting masked words, sentences, or segments from the given context. As shown in Figure 8, through extensive pre-training, LLMs encode linguistic knowledge, contextual comprehension, syntactic and semantic nuances, as well as general world knowledge directly into the model's internal parameters, specifically within the self-attention mechanism (query, key, and value vectors) and FFN layers (weights and biases).

Note that training LLMs requires intensive computation and significant parallel processing across many GPUs [25, 179]. These demands increase further with optimization techniques like mixed-precision arithmetic, distributed training across multiple nodes, and complex gradient synchronization. The parallel structure of Transformers addresses these issues by handling large parameter sets and extensive datasets across multiple GPU clusters in parallel. In contrast to the previous sequential models limited by recurrent operations, Transformers fully utilize modern parallel processing hardware, enabling scalable training. However, large-scale training using many GPUs typically takes weeks or even months of continuous operation [180–182]. *This extended training time arises from large memory requirements and frequent data synchronization among GPUs*, highlighting key infrastructure challenges related to scalability, memory capacity, and interconnect design.

**Ensuring coherence and generalization.** Many modern LLMs adopt an *auto-regressive* approach during training and inference [18, 176, 183]. The auto-regressive method sequentially predicts each token based solely on previously generated tokens, capturing the inherent sequential dependencies in linguistic data. Specifically, when generating a sentence, the model determines each word using only prior context, without incorporating information from subsequent tokens. This characteristic enables the auto-regressive approach to maintain logical coherence and contextual accuracy even in the absence of future context.

Figure 9a illustrates the training process, in which the model learns to predict the next token based on preceding tokens within the provided sequences. This sequential approach helps the model learn patterns in grammar, syntax, and logical progression, modeling temporal dependencies and maintaining coherence within generated texts. During inference, as shown in Figure 9b, auto-regressive models generate tokens one at a time. Each new token then becomes part of the context for subsequent predictions. Although this sequential mechanism




(a) Training with ground-truth tokens.

(b) Inference using previously generated tokens.

Figure 9: Auto-regressive model workflow.

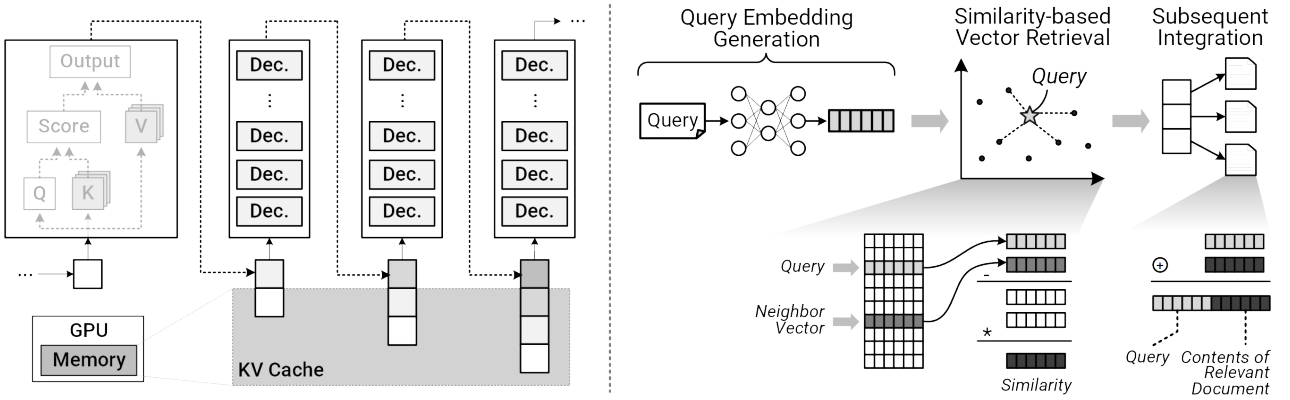
ensures coherent and contextually consistent outputs, it limits parallel computation, reducing inference speed relative to parallelized generation methods.

Despite these computational constraints, auto-regression remains widely used due to its proven capability to represent complex linguistic dependencies and produce accurate outputs, essential for high-quality language generation tasks. To mitigate inference limitations, modern LLMs also emphasize their extensive pre-training and strong generalization capabilities. By learning from large-scale and diverse textual datasets, these models form comprehensive internal language representations, enabling flexible application to various downstream tasks, often requiring minimal or no additional specialized training in a scenario known as zero-shot or few-shot learning [106, 184, 185]. Such generalization expands the applicability of LLMs beyond traditional language tasks into multimodal areas, including image generation, video synthesis, audio processing, and interactive conversational systems.

**Reducing redundancy and improving reliability in LLM inference.** As LLMs have become widely deployed across various applications, two critical techniques, *Key-Value* (KV) caching [3, 73, 74, 186] and *Retrieval-Augmented Generation* (RAG) [41, 42, 69], have been developed to overcome significant computational and accuracy-related challenges in LLM inference processes.

KV caching addresses computational inefficiencies inherent in the self-attention mechanism of LLMs. Due to the auto-regressive inference approach, where each token generation depends on previously generated tokens, LLMs must repeatedly calculate self-attention scores involving all previously processed tokens at every generation step. Without optimization, this results in redundant and repeated calculations, slowing inference speed. As shown in Figure 10a, KV caching resolves this by storing the computed attention scores as key-value pairs directly in GPU memory after their initial calculation. Once stored, these cached results can be directly reused for subsequent inference steps without additional computation. This can reduce redundant computational overhead and accelerate the inference process, particularly for longer input sequences. However, this efficiency gain comes at the cost of increased memory demands. Depending on the model size, token length, and inference complexity, KV caching can occupy between 30% and 85% of the available GPU memory [26–28, 187], considerably intensifying memory utilization and often surpassing the capacity of individual GPU modules.

On the other hand, RAG targets the inherent limitation of LLMs known as model hallucinations [188, 189], which are situations in which models produce plausible yet factually incorrect or contextually irrelevant outputs. Such inaccuracies arise because LLMs rely exclusively on internal knowledge learned during their training phase, lacking real-time or updated external context. RAG enhances model reliability by incorporating external knowledge retrieval directly into the inference workflow. When an input query is received, a RAG-equipped LLM first searches an external knowledge database, implemented as a specialized vector database [190–192] or retrieval system [193–195], for contextually relevant information (cf. Figure 10b). This retrieved context is then combined with the original input query, providing the model with accurate, externally verified information from




(a) Caching key-value pairs in GPU memory.

(b) Searching external database.

Figure 10: Inference Optimization Techniques.

which to generate its final response. Although this can reduce hallucinations and improves factual correctness, it introduces additional computational steps, including query embedding generation, similarity-based vector retrieval, and subsequent integration of retrieved information into the inference process. Consequently, RAG imposes computational complexity and requires significant memory capacity for maintaining large-scale vector databases. Moreover, network latency and bandwidth become critical performance factors, as rapid and reliable retrieval from external sources impacts response accuracy and inference latency.

Note that KV caching and RAG are all essential to address the crucial bottlenecks within LLM inference. While KV caching optimizes inference speed by minimizing redundant computations within the self-attention mechanism, RAG enhances output reliability and accuracy by leveraging external knowledge sources. Despite of these advantages, *deploying these inferencing techniques with LLM amplifies demands on GPU memory, computational resources, network bandwidth, and storage infrastructure*, further emphasizing the necessity for highly scalable and composable data center architectures capable of supporting diverse and intensive LLM workload requirements.

# 3. Scaling LLMs: Multi-Accelerator and Data Center Deployments

Modern data centers have evolved to handle diverse AI workloads, including recommendation systems [196–198], ranking algorithms [199, 200], and vision models [201, 202]. However, LLM workloads uniquely stress infrastructure due to their intense memory and communication needs. In this section we first examine how fundamental LLM concepts map onto multi-accelerator systems. We then analyze architectural and modular strategies adopted by contemporary data centers utilizing thousands of GPUs or accelerators. Throughout this discussion, GPU and accelerator terms are used interchangeably.

At the end of this section, we discuss the limitations of tightly integrated CPU-GPU architectures, which restrict scalability, flexibility, and efficient resource utilization. Addressing these constraints to meet large-scale AI demands necessitates adopting modular, independently scalable designs for CPUs, GPUs, memory, and networking components.

## 3.1. Mapping and Challenges of Deploying LLMs on Multi-Accelerator Systems

The exponential scaling of modern LLMs, particularly those based on Transformer architectures, surpasses the memory and computational capabilities of individual GPUs [43, 44, 203]. This necessitates distributing large models across multiple GPUs. Each GPU within these multi-GPU setups manages specific subsets of parameters and computational tasks, enabling effective parallelization and distributed training.

**Multi-GPU LLM training: Model partitioning, parallelization, and overheads.** The primary challenge in multi-GPU LLM training is efficiently partitioning and synchronizing extensive model parameters, activations, and gradients across GPUs, ensuring coherent and effective distributed computation.

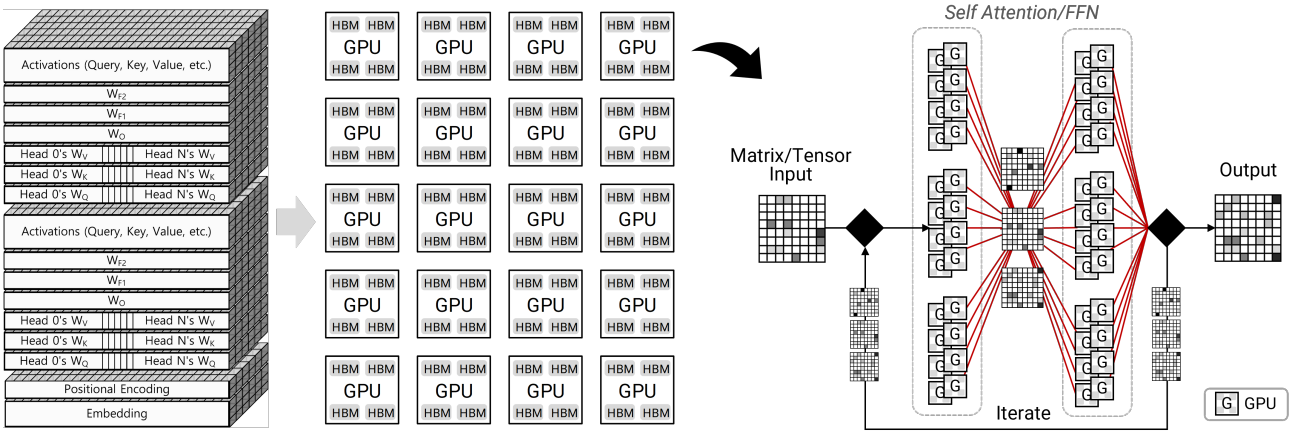
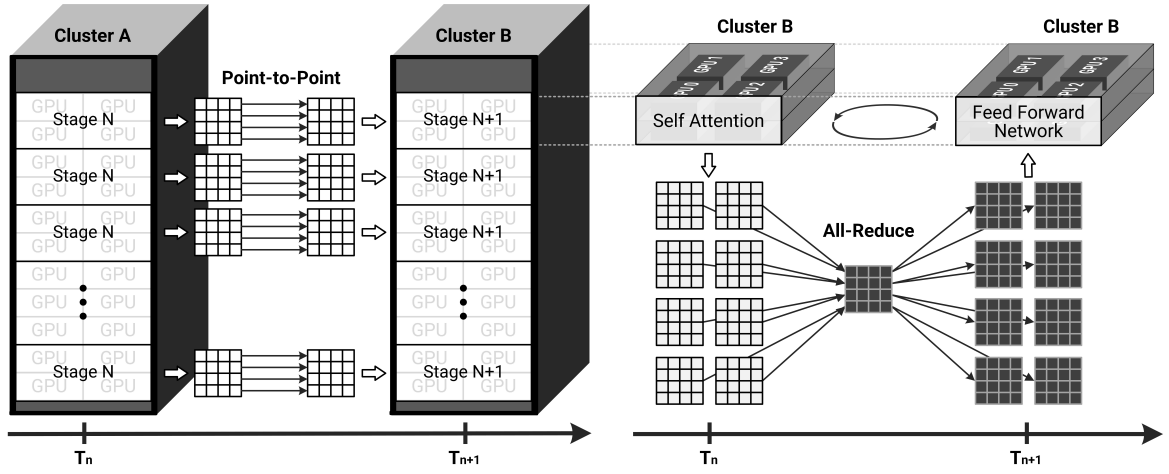



Figure 11: Transformer-based model partitioning and synchronization overhead.

Figure 11 illustrates the partitioning strategy and tensor synchronization across GPUs in Transformer-based models, emphasizing the frequent exchange of outputs from both self-attention and FFN computations. The self-attention mechanism within Transformer architectures computes interactions across all tokens in a sequence. At first glance, it may appear that GPUs must generate and exchange partial query, key, and value vectors. However, each GPU can independently compute self-attention using only its assigned partial vectors, enabled by multi-head attention and grouped query attention, which partition one large attention layer into multiple parallel smaller layers. Nonetheless, periodic synchronization of these computed vectors and their gradients remains necessary to ensure global coherence and consistency across the distributed architecture [26, 204, 205]. This synchronization step is essential for accurate gradient computation and parameter updates, significantly increasing demands on inter-GPU communication bandwidth and memory resources.

In addition to self-attention, Transformer architectures include FFN layers, which facilitate independent token-level computations. Although FFN operations allow parallel execution, exchanging intermediate results and synchronizing gradients across GPUs remain required during forward and backward passes [206–210]. Such gradient synchronization necessitates frequent exchanges of intermediate gradient updates [211–213], further elevating inter-GPU communication overhead.

Advanced parallelization techniques, such as *pipeline parallelism* and *tensor parallelism* [43, 45, 179, 214, 215], also play crucial roles in distributed LLM training. Figure 12a visualizes pipeline parallelism by illustrating the sequential "stages" of model execution, showing how each GPU cluster handles specific layers of the Transformer




(a) Pipeline parallelism (PP).

(b) Tensor parallelism (TP).

Figure 12: Pipeline parallelism (PP) and tensor parallelism (TP) for transformer architecture.

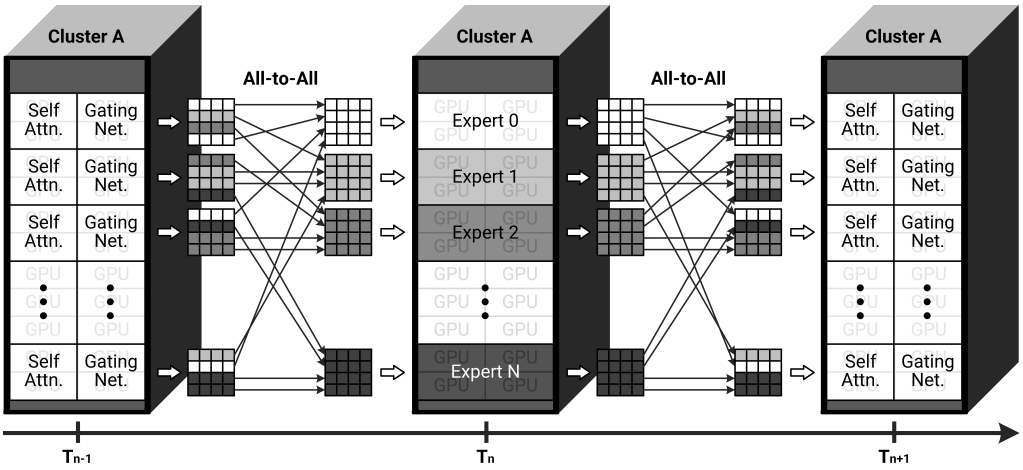


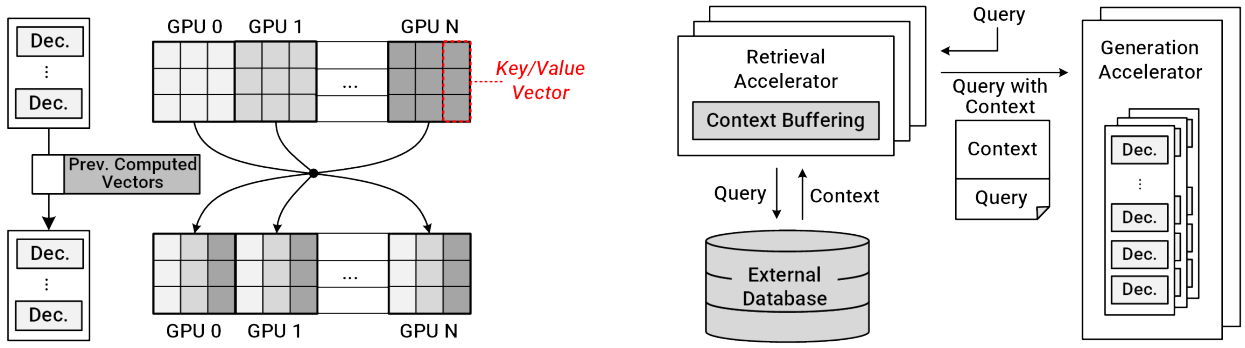

Figure 13: Expert parallelism (EP) for mixture of experts (MoE) architecture.

model to optimize resource utilization. Pipeline parallelism divides the Transformer model into sequential stages, with each stage processed on separate GPU clusters. Although this method increases available parallel computation, careful orchestration is required to minimize idle periods (pipeline bubbles [45, 179, 216–218]) caused by inter-stage data dependencies. Stages must synchronize their data handoffs to ensure smooth operation and maximum utilization of GPU resources. Pipeline parallelism is particularly effective for large models where the computation within each stage can fully utilize individual GPUs [25, 43, 179].

On the other hand, tensor parallelism complements pipeline parallelism by partitioning large tensor operations, such as matrix multiplications, across multiple GPUs. This approach enables simultaneous computation within layers, accelerating the processing of large tensor operations. However, tensor parallelism requires frequent synchronization of partial results across GPUs, typically using *collective communication* operations such as All-Reduce, All-Gather, and Reduce-Scatter [29, 53, 219–222]. These collective communication operations enable GPUs to exchange and aggregate intermediate computational results, maintaining consistency across parallel computations. Figure 12b further illustrates tensor parallelism by depicting how large tensor computations are distributed across GPUs, highlighting the critical role of collective communication operations in synchronizing partial computations. Efficient implementation of tensor parallelism thus relies on optimized collective communication algorithms and high-performance interconnect infrastructures to maintain low communication latency and high throughput.

Furthermore, the adoption of dynamic computational strategies, exemplified by MoE architectures, adds complexity to training workflows [171, 223]. Figure 13 shows the distribution of MoE expert modules across GPUs, emphasizing how each GPU independently manages distinct subsets of data computations. MoE models partition the network into multiple expert modules, each hosted on dedicated GPUs or GPU clusters. Each GPU acts as an independent expert performing distinct forward and backward computations for specific input data subsets [168, 224–226]. Input data sequences, represented tokens, are distributed across multiple GPUs based on predetermined criteria or routing strategies. Tokens or token segments representing parts of sentences or queries are allocated to GPUs according to the model's expert selection policy. After tokens are assigned, each GPU expert processes its designated computations individually. However, Transformer-based models require aggregating outputs from multiple experts to generate meaningful predictions, leading to "frequent exchanges" of intermediate results among GPUs. The figure illustrates the aggregation and synchronization process among GPUs in MoE architectures, showcasing the intensive exchange of intermediate computational results required for maintaining global model consistency. Regular aggregation and synchronization of gradients across experts are essential to maintain global model consistency. Consequently, *MoE training escalates inter-GPU communication demands*, requiring sophisticated high-bandwidth, low-latency interconnect infrastructures.

**Multi-GPU LLM inference: Optimization techniques and challenges.** In contrast to the training phase, inference is primarily known to emphasize computational speed and real-time responsiveness [227–229]. However, recent optimization techniques [230–232] for inference workloads have shifted part of the performance emphasis from pure computation toward increased memory capacity and inter-GPU communication bandwidth. These shifts result from advanced methods designed to reduce redundant calculations and enhance contextual accuracy, introducing substantial new system-level challenges.




(a) KV caching for token reuse across GPUs.

(b) RAG execution with large-scale retrieval integration.

Figure 14: Communication and memory overhead in KV caching and RAG inference.

Figures 14a and 14b show inter-GPU communication and external data access patterns for inference optimizations such as KV caching and RAG, respectively; as shown in Figure 14a, KV caching exemplifies this shift by storing previously computed key and value vectors directly within GPU memory. While this technique reduces redundant computation, leading to significantly faster inference, it substantially elevates GPU memory demands [30, 75]. As models and context windows scale, KV caches become massive, often requiring careful partitioning and frequent synchronization across GPUs [74, 233]. As a result, this places intensive pressure on memory management strategies and dramatically increases inter-GPU communication overhead to maintain cache coherence and efficiency.

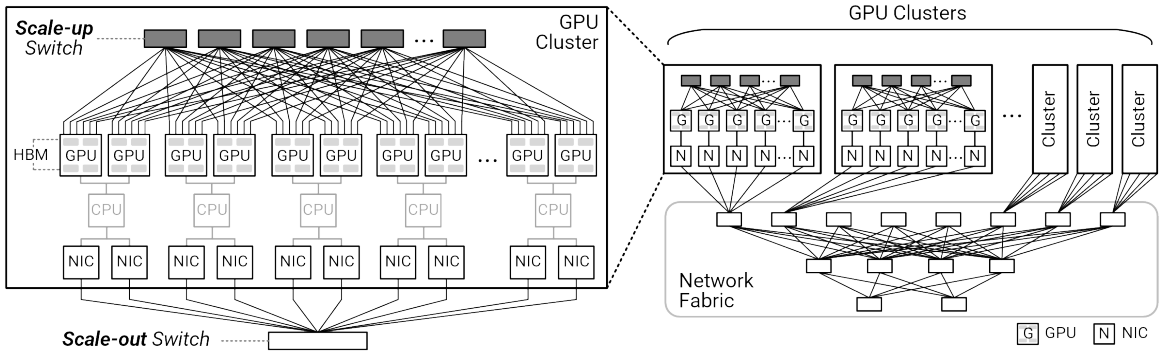
On the other hand, as illustrated in Figure 14b, RAG further intensifies demands on both memory and communication resources. By incorporating external knowledge bases into inference, RAG improves the accuracy and contextual relevance of outputs [68, 234–236]. However, it requires GPUs to execute rapid, frequent queries to external databases, swiftly retrieve relevant data, and seamlessly integrate this external information into ongoing computations. These operations drastically increase memory usage to temporarily store the retrieved data, and place heightened demands on network bandwidth and low-latency communication infrastructure [41, 68, 237], complicating system design and performance optimization.

Note that auto-regressive inference methods also impose additional memory and interconnect-related constraints. Due to the sequential dependency inherent in token generation, each prediction explicitly relies on previously generated tokens, severely restricting parallel execution. Consequently, GPUs must exchange intermediate computational results as soon as possible and maintain synchronization across inference steps. This sequential dependency not only limits achievable parallelism but also escalates inter-GPU communication traffic, thereby intensifying demands for low-latency and high-throughput network connectivity and sophisticated memory management solutions to mitigate GPU idle times.

Considering all these factors, both training and inference workloads of modern LLMs increasingly emphasize memory capacity and inter-GPU communication infrastructure. Training requires sophisticated model partitioning, frequent synchronization, and advanced parallelization techniques due to parameter and activation sizes exceeding GPU memory capacities. Similarly, inference optimizations such as KV caching, RAG, and auto-regressive methods reduce redundant computations but further amplify memory usage and inter-GPU synchronization overhead. Therefore, modern AI infrastructures must adopt flexible architectures that efficiently manage memory and communication resources in addition to computational performance, addressing the comprehensive needs of contemporary LLM workloads.

## 3.2. Scaling Multi-Accelerator Systems: Scale-up and Scale-out

Addressing inter-GPU communication challenges in multi-GPU training and inference requires sophisticated hardware interconnect and network solutions tailored to varying performance and scalability requirements. Two primary architectural strategies, *scale-up* and *scale-out*, are commonly adopted to handle these different operational scenarios. Generally, scale-up architectures utilize high-speed "interconnects" such as NVLink [79, 80], NVLink Fusion [82, 83], UALink [76, 77], and CXL, while scale-out architectures employ high-bandwidth, "long-distance networks" like Ethernet [238–240] or InfiniBand [241–243].




(a) Scale-up and scale-out strategy for multi-gpu deployment. (b) Interconnected GPU clusters via network fabric.

Figure 15: Scale-up vs. scale-out and networked GPUs.

**Scale-up architecture: High-speed direct interconnect.** Figure 15a compares scale-up and scale-out GPU interconnect architectures for multi-GPU deployments. The scale-up strategy tightly couples a limited number of GPUs using specialized high-speed, accelerator-centric interconnects. These direct, high-bandwidth connections optimize data transfer efficiency, which is particularly beneficial for workloads involving frequent and intensive data exchanges within closely integrated GPU clusters. Scale-up solutions are advantageous for tasks requiring maximum intra-node communication speed and minimal latency, thereby significantly improving performance for computationally intensive scenarios such as training, real-time inference, and GPU-heavy AI operations.

Prior to the emergence of LLMs and the latest generation of data center architectures, the number of GPUs requiring these closely integrated direct interconnects was limited. However, as models grow more complex and data volumes increase significantly, the number of GPUs needing such connectivity is rising exponentially. Furthermore, many optimization techniques for LLMs now require high-speed, low-latency data exchanges and consistent I/O data sharing. Consequently, there is a growing trend in modern data centers to deploy more GPUs per rack, aiming to enhance computational efficiency, reduce communication overhead, and lower the total cost of ownership (TCO). For example, NVIDIA has introduced a new architecture, featuring compact and liquid-cooled node units designed specifically for high-density GPU deployments. These nodes utilize high-speed NVLink and NVSwitch interconnects to tightly integrate up to 72 GPUs per rack. This type of direct interconnect designs enhances computational throughput, optimizes inter-GPU communication efficiency, and improves thermal management, thereby reducing operational complexity and lowering the TCO.

**Scale-out architecture: Long-distance network interface.** In contrast, the scale-out approach is designed for extensive, data center-scale deployments that may involve thousands of GPUs distributed across multiple racks or nodes. This strategy enables broader scalability and more flexible resource management. Scale-out architectures mainly utilize "long-distance" network-based fabrics, relying on *network interface cards* (NICs) and RDMA-enabled communication protocols for GPU-to-GPU interactions [244–246]. As illustrated in Figure 15b, GPUs are organized into clusters that are interconnected via network fabrics, allowing for modular and dynamic system configurations.

While long-distance network fabrics provide excellent scalability, flexibility, and potentially higher aggregate bandwidth, they unfortunately introduce additional overhead. This overhead arises from complex hardware designs, sophisticated network protocols, and software-mediated communications. Specifically, data serialization and deserialization, network protocol processing, and software-level interactions significantly increase communication latency compared to tightly coupled, hardware-based scale-up architectures. Therefore, carefully evaluating these trade-offs between scale-up and scale-out architectures is essential when designing large-scale, distributed AI systems to ensure performance and efficiency objectives are effectively achieved.

On the other hand, CPUs still play a crucial supporting role in GPU-centric AI infrastructures for both scale-up and scale-out systems. While GPUs handle primary computational tasks, CPUs provide system orchestration capabilities, managing GPU coordination, data transfers, and network interfaces. Each GPU or GPU cluster thus integrates one or more CPUs and NICs as fundamental components. In the past, there have been various approaches toward resource disaggregation, similar to those presented in this technical report. However, com-

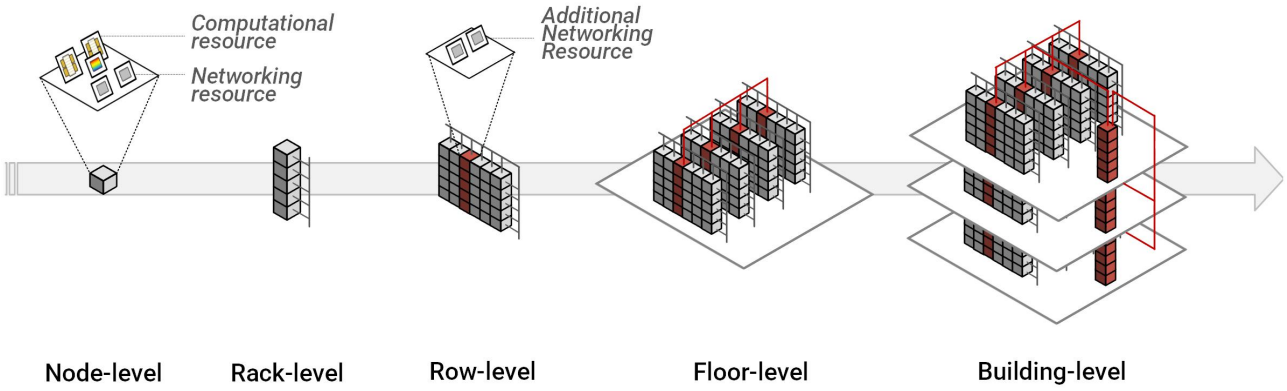



Figure 16: Hierarchical data center architecture.

plete physical resource disaggregation has not been fully realized because CPUs should act as host processors, managing the bus interfaces and memory controllers that interface directly with GPUs or accelerators. Instead, recent industry trends emphasize tighter integration within single nodes or adopt modular scaling methods. An illustrative example of such node-level integration is NVIDIA's latest GPU model (Grace Blackwell), which will be further examined in the subsequent subsection.

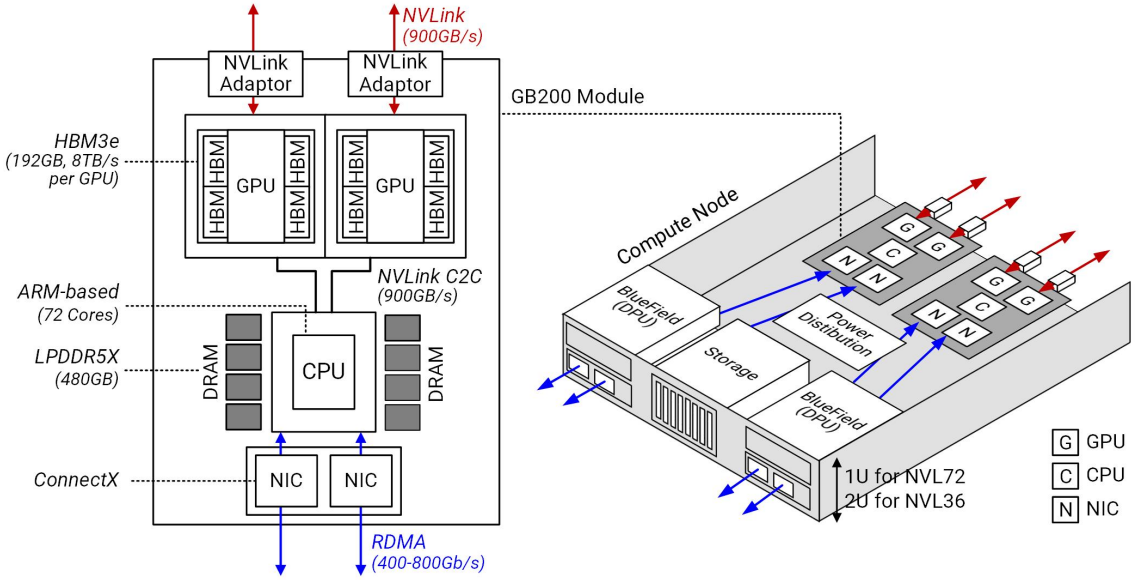
## 3.3. Large-Scale AI Infrastructure: Hierarchical Data Center Architecture with Blackwell Instances

Real-world data centers in practice adopt hierarchical architectures to support diverse workloads and varying infrastructure demands. As shown in Figure 16, these hierarchical designs integrate computational and networking resources across multiple abstraction levels, which are namely, *nodes*, *racks*, *rows*, *floors*, and entire *buildings* [247–251]. Specifically, node-level components form the foundational computing units, which are further aggregated into racks to enhance computational density and interconnectivity. Subsequently, racks are organized into rows and then systematically expanded into floor-level structures, forming integrated building-scale deployments.

In the following subsections, we detail each hierarchical level, starting with the fundamental node configurations and gradually progressing toward comprehensive building-level integration. To illustrate practical node configurations, we reference NVIDIA's recent Grace Blackwell architecture [31, 39, 47–49, 252] as a representative example of contemporary designs. The Blackwell architecture introduces notable hardware enhancements, such as increased *high-bandwidth memory* (HBM [35–37, 253]) capacity per GPU and closer CPU-GPU integration, directly addressing key challenges including memory management, inter-GPU communication, and computational coordination. While Blackwell provides specific context, our primary emphasis remains on the broader understanding of general data center architectures, which highlights their respective advantages and limitations.This structured discussion lays the groundwork for exploring advanced designs in the following sections.

**Node- and rack-level configuration: Hierarchical integration of CPU-GPU modules.** The fundamental building block of modern AI data centers is the "compute node", an integrated computational unit comprising CPUs, GPUs, memory, and network interfaces. Figure 17a illustrates a representative node setup using the GB200 configuration as an example. Specifically, each GB200 integrates one high-performance ARM-based CPU with 72 cores and two GPUs, tightly coupled via NVLink *chip-to-chip* (C2C) interface [33, 254]. Each GPU contains approximately 192 GB of HBM3e, delivering bandwidths up to 8 TB/s per GPU [252, 255], supporting large-scale AI models and inference workloads. In addition, the CPU provides up to 480 GB of LPDDR5X DRAM [255, 256], which maintains low-latency and unified memory communication with GPUs via NVLink C2C, achieving approximately 900 GB/s of bandwidth [33, 254, 257]. This tightly integrated design results in a memory-unified computational domain within each GB200.

Each compute node also integrates advanced NICs to enable high-speed connectivity with external network fabrics. Common examples of the NICs include NVIDIA's BlueField data processing units (DPUs) [258] and ConnectX adapters [259, 260], which are directly installed within individual nodes. These NICs provide hardware

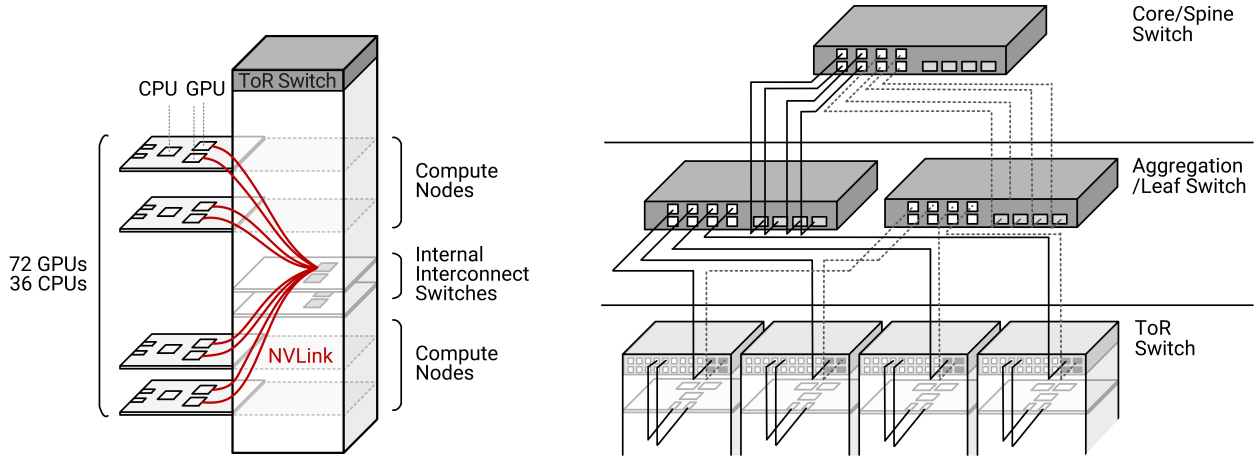



(a) Blackwell architecture (GB200 module). (b) A compute node setup with two GB200 modules.

Figure 17: Node-level configuration of GB200.

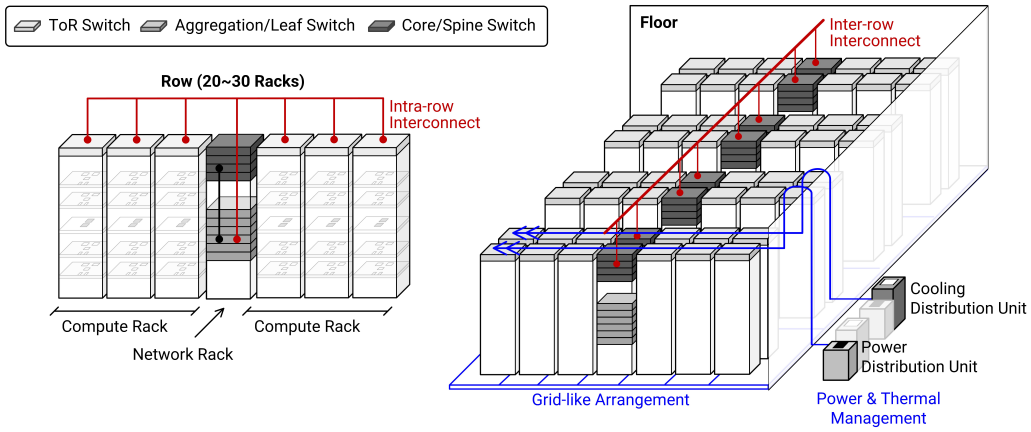
acceleration for networking functions, support high-bandwidth communication (typically 400 to 800 Gb/s per node), and enable RDMA capabilities for high-throughput data transfers. The integration of such NICs within a node ensures that every computational unit can participate in data center-scale networking, supporting both internal cluster operations and communication with broader network fabrics. Note that, as shown in Figure 17b, a compute node can contain two GB200 modules (two CPUs and four GPUs), packaged into compact 1U or 2U form factors optimized for dense rack-level deployments [39, 261].

At the rack level, multiple compute nodes aggregate into high-density computational clusters. Figure 18a illustrates a representative rack-level architecture, using the GB200-based configuration as a concrete example to clarify typical design choices. In this example, nodes interconnect via rack-scale internal interconnect fabrics (e.g., NVLink and UALink). A standard rack can accommodate up to 36 GB200 modules, collectively comprising 72 GPUs and 36 CPUs [39, 47]. Internally, all GPUs within the rack are interconnected using dedicated




(a) Internal connectivity through NVSwitch. (b) External connectivity through ToR switch.

Figure 18: Rack-level configuration of NVL72.




(a) Row-level configuration.

(b) Floor-level configuration.

Figure 19: Row and floor-level configuration.

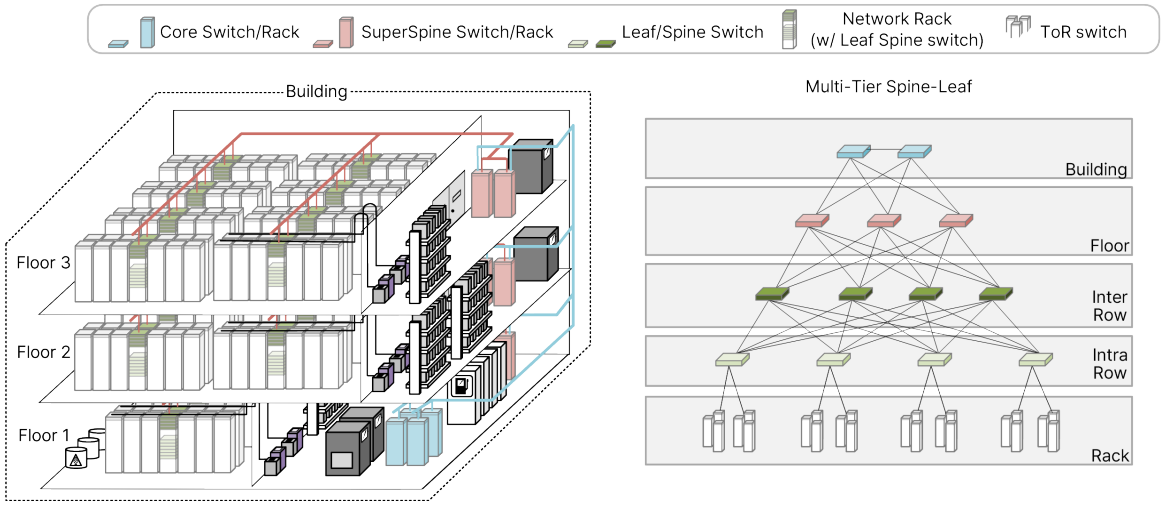
internal interconnect switches (e.g., NVSwitches [262, 263]) carefully distributed across the rack. Such internal interconnects provide high-bandwidth, low-latency communication, ideal for workloads requiring intensive intra-rack data transfers such as large-scale model training and real-time inference tasks. This internal connectivity thus creates an efficient, "scale-up domain" within each rack.

Simultaneously, as illustrated in Figure 18b, each node within the rack connects directly to *top-of-rack* (ToR) network switches. The ToR switches aggregate traffic from all internal nodes and manage external connectivity, typically at bandwidths ranging from 400 to 800 Gb/s. ToR switches are positioned at the rack's upper portion, managing node-level network communication, reducing cable complexity, and minimizing latency for external network interactions. By simplifying cable management, this configuration can improve operational efficiency and scalability.

To enable the seamless expansion of data center infrastructure beyond the confines of a single rack, ToR switches are equipped with uplink ports that connect to higher-level, long-distance network infrastructures, such as aggregation switches [264, 265] or leaf switches [266, 267] within a spine-leaf topology [267–269]. Through these hierarchical connections, ToR switches play a pivotal part in scaling networks to encompass multiple racks, rows, floors, and ultimately entire buildings, as will be discussed shortly. Note that ToR switches not only facilitate immediate rack-level communication but also act as the bridge to higher tiers of the data center network. Thus, ToR switches serve a dual role: aggregating and optimizing internal rack-level communication and providing flexible, high-bandwidth connectivity for scalable expansion across larger data center deployments. The combined use of intra-rack interconnects such as NVLink and external network connectivity via ToR switches forms a critical infrastructure in modern AI system design, simultaneously addressing GPU performance and scalability demands, yet also introducing inherent scalability limitations.

**Row and floor-level configurations: Scaling infrastructure.** In modern data center architectures, dedicated network racks, distinct from compute racks, serve as centralized aggregation points within each row. These network racks house aggregation switches or spine-leaf switches that interconnect the ToR switches of multiple compute racks in the same row, forming the backbone of intra-row communication. Figure 19a illustrates a typical row-level configuration in modern data centers, showing multiple compute racks connected to these network racks via InfiniBand or Ethernet switches [270, 271]. Each row consists of several compute racks, each containing densely packed compute nodes, and one or more centrally positioned network racks dedicated to switching and aggregation.

As shown in the figure, the network racks located within the row (often at the center or end) are also populated with high-bandwidth switches, such as InfiniBand (Quantum-2) [272] or Ethernet (Spectrum-X) [273], which aggregate and route traffic between all compute racks in the row. These switches support extensive bandwidth capacities, typically ranging from 200 to 800 Gb/s per port, enabling efficient data exchanges among the compute racks. By centralizing the switching infrastructure within dedicated network racks, cable management is simplified, network latency is minimized, and scalability is enhanced, as additional compute racks can be seamlessly added and interconnected through the aggregation switches.




(a) Building-level configuration.

(b) Multi-tier spine-leaf data center topology.

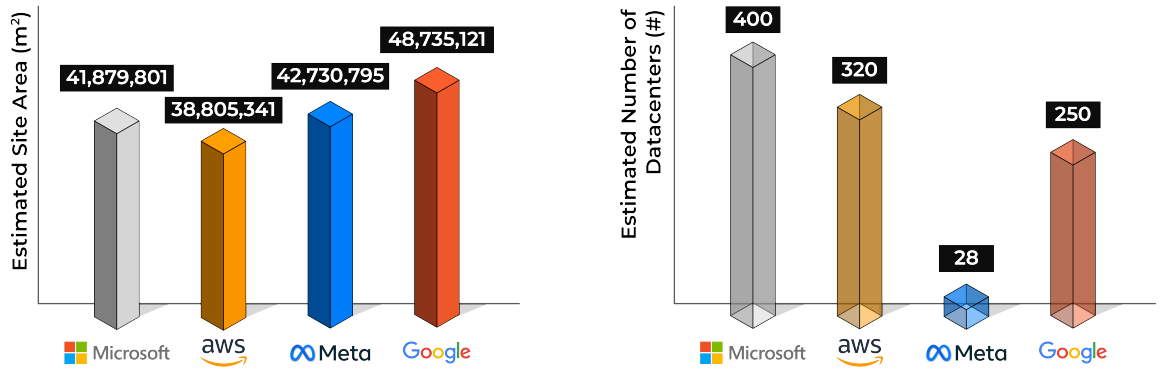
Figure 20: Overall data center topology and building configuration.

While such row-level communication structures have assisted in scaling data centers by aggregating existing nodes and racks, recent operational environments handling large models and extensive data, such as LLMs, introduce several structural limitations, particularly related to high-speed inter-GPU synchronization. As the frequent synchronization required among GPUs significantly reduces GPU utilization, strategic infrastructure planning at the row level is becoming increasingly critical for ensuring the stability and performance of large-scale AI workloads. In this technical report, *we emphasize the importance of both inter-row and intra-row communications*, which are currently managed by scale-out architectures. However, *this emphasis also sets the stage for further exploration into potential enhancements through scale-up architectures.*

On the other hand, the transition from row-level to floor-level configuration represents a critical step in scaling data center infrastructure. Specifically, optimizations at the row-level (e.g., efficient intra-row networking, organized cable management, and streamlined thermal strategies) must be cohesively integrated across multiple rows at the floor-level. Typically, floor-level configurations interconnect multiple rows, forming grid-like layouts to optimize spatial efficiency and inter-row connectivity [248, 274]. For instance, a single floor generally aggregates several rows, each containing approximately 20 to 30 racks, resulting in coordinated management of hundreds of racks. Figure 19b illustrates such a floor-level arrangement, emphasizing the coordinated interconnections between rows to facilitate efficient data flow and resource sharing. At this scale, spatial organization, efficient inter-row networking, and comprehensive infrastructure management become crucial. In addition, careful design considerations for scale-up and scale-out domains, based on the characteristics of specific interconnect and network technologies, are essential. Due to the hardware layout characteristics of data centers, frequent inter-row and intra-row communications occur, driven primarily by synchronization requirements for parallel GPU and accelerator processing or memory access data exchanges. Currently, however, most of these row-level communications rely on scale-out domain connections, resulting in relatively high overhead during data transfers. In a subsequent section, we will discuss various strategies to enhance scale-up architectures, enabling low-latency, high-speed intra-row and inter-row communications.

Note that advanced thermal and power management solutions, such as liquid-cooling distribution units [275, 276] and power distribution units [277–279] integrated at the row or rack level, are also important for dissipating heat from densely packed computing nodes. These strategies collectively ensure stable operating conditions, minimize performance degradation, and enhance the overall reliability and efficiency of large-scale AI infrastructure deployments.

**Building-level integration: From AI infrastructure formation to campus-scale.** As illustrated in Figure 20a, building-level integration represents the highest tier of hierarchical data center organization, interconnecting multiple floors, each composed of interconnected rows and racks, into unified, large-scale AI infrastructure. At this scale, managing coherent resource allocation, data movement, and network coordination across thousands to tens of thousands of GPUs distributed over multiple floors poses significant operational




(a) US site area sum per hyperscaler.

(b) Number of hyperscaler's data centers.

Figure 21: Hyperscaler's site area and data center count.

complexities. For instance, modern large-scale deployments often connect multiple floors using long-distance network technologies, logically enabling extensive GPU integration across buildings [280–282].

However, scaling infrastructure to the building-level introduces unique challenges beyond those encountered at lower hierarchical levels. Specifically, communication between floors increases network latency and congestion, often limiting practical GPU utilization to approximately half of the theoretical peak performance [32, 247, 283]. In typical, building-level integration employs hierarchical network topologies, such as multi-tier spine-leaf or multi-level fat-tree architectures, interconnecting multiple floors to balance communication load, reduce latency, and manage congestion effectively (cf. Figure 20b). Moreover, handling power distribution, thermal conditions, and fault tolerance across multiple floors presents additional difficulties that further complicate operational efficiency [284, 285]. To mitigate these challenges, automated monitoring and centralized resource management systems become indispensable, providing real-time visibility into computational loads, network performance, thermal management, and hardware status.

Unfortunately, despite these sophisticated management strategies, fundamental communication bottlenecks, such as inter-GPU synchronization overhead and extensive data movement across hierarchical layers, remain structurally unavoidable. These persistent communication challenges structurally constrain scalability and efficiency. Addressing these communication constraints therefore requires a new type of interconnect architectures and composable system designs, which will be explored in depth in the following sections.

Note that multiple building-level structures collectively form campus-scale infrastructures that support large-scale data center deployments. For context, Figures 21a and 21b illustrate the current scale of data centers operated by major hyperscalers, including Microsoft, Meta, Google, and Amazon, in response to growing demand for AI infrastructure. Figure 21a shows the total site area of U.S.-based data centers for each company, including planned facilities projected to be completed by 2027 [286]. Figure 21b presents the number of data centers as defined by each hyperscaler [287–291].

To convey the scale of these deployments, Meta's total site area reaches approximately 42 million $m^2$, equivalent to about 5,300 standard soccer fields. Microsoft operates nearly 400 data centers worldwide, while AWS and Google manage between 200 and 300 centers. Meta maintains around 30 centers, but each facility is significantly larger in area, resulting in a total site footprint comparable to that of the other hyperscalers. Meta's infrastructure emphasizes large-scale, high-density design, prioritizing capacity and operational efficiency. These differences reflect varied infrastructure scaling strategies, which in turn influence the architectural complexity and efficiency of each hyperscaler's deployment model.

## 3.4. Constraints and Challenges in GPU-Integrated AI Infrastructure

**Beyond compute: Why GPU parallelization faces fundamental constraints.** So far, we have discussed hierarchical data center architectures exemplified by the Blackwell configurations, which scale computational resources from nodes to entire buildings, integrating thousands to tens of thousands of GPUs. However, despite such architectural scalability, GPU parallelization encounters intrinsic constraints caused by memory limitations and unavoidable communication overheads. As discussed previously, modern LLM workloads exceed the memory capacities of individual GPUs due to extensive model parameters, large intermediate states generated

by attention mechanisms, and sizable activation data. As a result, partitioning computations across multiple GPUs is unavoidable, but each parallelization approach introduces critical performance trade-offs related to inter-GPU synchronization, communication overhead, and operational complexity.

Specifically, model parallelism distributes model parameters across multiple GPUs, addressing memory capacity constraints but introducing frequent and intensive inter-GPU synchronization overhead. Similarly, data parallelism duplicates the entire model across GPUs for parallel batch processing, yet the collective synchronization operations (e.g., All-Reduce) impose significant communication overhead, limiting GPU utilization to approximately 35–40% of theoretical peak performance [292–297]. Pipeline parallelism segments models sequentially across GPUs to accommodate larger models, but inter-stage data transfers cause pipeline bubbles (GPU idle periods), restricting utilization to about 50% [25, 45, 179, 216, 298]. Even hybrid approaches, combining multiple parallelization strategies, cannot fully mitigate the inherent communication overhead intrinsic to multi-GPU environments.

Critically, these structural bottlenecks persist despite advances in modern hardware architectures. While the hardware innovations (featuring high-bandwidth NVLink, increased HBM3e memory capacities, and integrated CPU-GPU designs) partly reduce communication latency and improve overall throughput, they cannot eliminate synchronization overhead and complexity in memory management, which are all intrinsic. Thus, *the structural limitations of existing GPU parallelization methods, rooted deeply in communication overhead and synchronization requirements, remain key challenges.* Addressing these intrinsic bottlenecks is essential for the future optimization and architectural evolution of AI infrastructure.

**Rethinking resource integration: Limitations of coupled CPU-GPU deployments at scale.** Tightly integrated CPU-GPU modules, as exemplified by the Blackwell architecture, provide performance benefits for specific computational tasks. However, deploying such tightly coupled resources across large-scale data centers presents critical constraints in terms of scalability, flexibility, and operational efficiency. Specifically, these integrated modules enforce rigid resource coupling, restricting the independent scalability of computational, network, and memory resources.

In large-scale AI environments, this tightly coupled approach exacerbates two primary challenges: *significant inter-node communication overhead* and *inflexible memory allocation*. Each CPU-GPU node requires dedicated network connections for inter-node data transfer and synchronization, causing network complexity and communication overhead to escalate linearly with system size. This direct, tightly coupled network topology notably increases latency and synchronization delays among GPUs, degrading performance, for communication-intensive tasks such as LLM training and inference.

In addition, the rigid CPU-to-GPU ratio dictated by integrated modules (e.g., one CPU per two GPUs in GB200/300) often results in underutilized CPUs as the infrastructure scales, leading to inefficient resource use and unnecessary operational costs. Moreover, fixed memory binding within these modules prevents independent memory scaling, forcing proportional increases in memory capacity with node additions. This inflexibility causes either severe memory underutilization or insufficient memory capacity, thus restricting the effective handling of varying AI workloads.

Lastly, tightly coupled CPU-GPU modules inherently complicate maintenance and upgrades. Component-level failures necessitate replacing entire modules, increasing downtime and operational expenses. Furthermore, integrated architectures limit the timely incorporation of technological advancements, as simultaneous CPU-GPU upgrades are typically required, reducing system agility and delaying modernization efforts.

These structural and operational constraints highlight the fundamental limitations of integrated CPU-GPU modules for meeting the dynamic demands of large-scale AI deployments. To address these critical bottlenecks, future data center architectures must adopt modular, independently scalable designs for CPUs, GPUs, memory, and network resources. Such disaggregated approaches will ensure scalability, operational flexibility, and efficient resource utilization at data center scale [299–303].

# 4. Leveraging CXL for Diverse AI Performance Metrics

Modern AI infrastructures, particularly those supporting large-scale LLM deployments, must concurrently satisfy multiple distinct performance metrics, including i) computational throughput, ii) model size, iii) memory bandwidth, iv) memory capacity, v) network bandwidth, vi) latency sensitivity, and vii) overall system scalability. As shown in Figure 22, these performance dimensions exhibit varying relative importance depending on specific operational scenarios, highlighting the dynamic and workload-specific nature of AI infrastructure demands [87, 304–308].

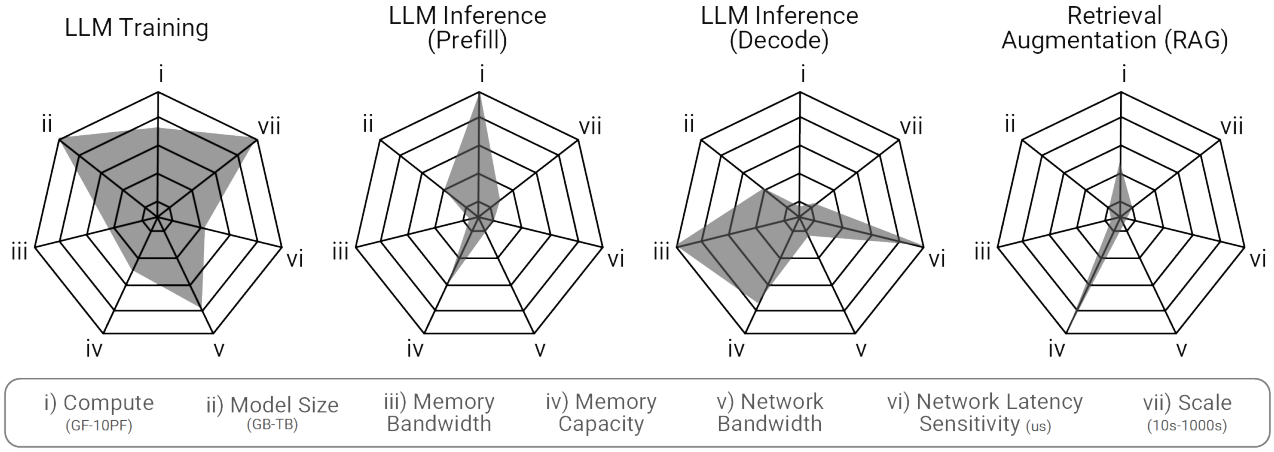



Figure 22: Relative importance of performance metrics across different operational scenarios.

To illustrate these varying requirements, we analyze distinct workload scenarios, including LLM training [309, 310], the prefill and decode phases of LLM inference [309, 311], and RAG workloads [70, 71, 312], each emphasizing different combinations of these performance metrics. This analysis reveals that no single architectural configuration can optimize all metrics across these diverse scenarios. Addressing these multidimensional and evolving performance demands necessitates modular, composable architectures capable of scaling computational, memory, and network resources.

We advocate that *Compute Express Link* (CXL) technology can address this requirement by enabling resource disaggregation and dynamic composability, thereby providing the flexibility necessary to adapt and efficiently scale resources according to specific AI workload characteristics. In this section, we first discuss the challenges of meeting multidimensional requirements in modern AI infrastructures. We then provide the background by reviewing the evolution of various CXL specifications, emphasizing their architectural developments and their implications for meeting diverse AI infrastructure requirements. Finally, we propose CXL-enabled tray designs and rack architectures tailored to these diverse AI infrastructure scenarios, enabling concurrent optimization of multiple performance metrics across dynamically evolving workloads.

## 4.1. Key Challenges in Simultaneously Optimizing Performance Metrics

To facilitate understanding, we categorize factors influencing the previously mentioned seven performance metrics into four primary groups based on the characteristics of different ML workloads. These groups are computational throughput, memory (capacity and bandwidth), network communication, and latency sensitivity. This subsection first explains why each of those groups is important in running the different workloads and then examines why conventional data center architectures have difficulty optimizing these interconnected dimensions in parallel.

**Computational throughput.** For computational throughput, modern LLM training requires extremely large model sizes, involving tens to hundreds of billions of parameters, to achieve adequate expressive power and superior generalization capabilities. Specifically, large models capture complex linguistic relationships and subtle contextual nuances, exhibiting emergent capabilities, such as advanced reasoning and improved context comprehension, which smaller models fail to deliver. For instance, models adopting MoE architectures significantly amplify parameter counts, often into the trillions, by selectively activating specialized expert networks per input token, thus achieving enhanced model capacity and performance. To manage and process these vast parameter sets within practically acceptable timeframes, computational throughput is the matter, and to get high computational throughput, substantial parallelization across thousands of GPUs becomes essential. However, as discussed earlier in Section 3.4, deploying large-scale GPU clusters introduces frequent synchronization events, particularly during collective operations, such as gradient aggregations and expert network activations unique to MoE structures. These intensive collective communication patterns escalate network bandwidth demands, creating critical performance bottlenecks. Therefore, practical GPU utilization in realistic training scenarios remains limited, achieving less than half of theoretical peak performance due to synchronization-induced overheads and network congestion [25, 179, 216, 294, 313, 314].

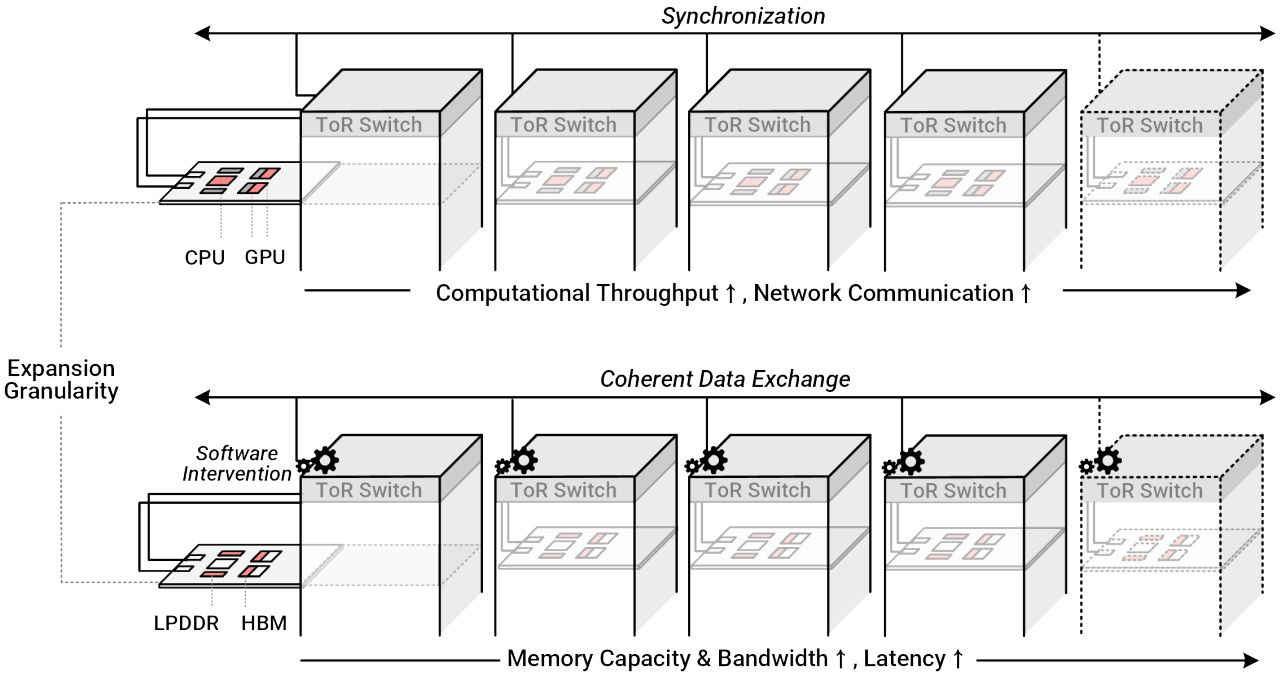



Figure 23: Competing constraints of performance dimensions

**Memory capacity and bandwidth.** The memory capacity and bandwidth are equally important and represent critical bottlenecks. The exponential growth of model parameters and intermediate activations in modern LLMs, reaching tens to hundreds of billions of parameters, results in memory requirements exceeding hundreds of TBs during training. In practice, model parameters, intermediate activations, optimizer states, gradient buffers, and related metadata must reside concurrently in memory, amplifying overall memory demands [21, 22, 315]. These extensive memory requirements considerably surpass the local GPU memory capacity, ranging from tens to a hundred GB in contemporary architectures. Hence, traditional architectures that tightly couple CPUs and GPUs with fixed memory configurations restrict independent scaling of memory resources, limiting their ability to accommodate massive, multi-hundred-TB-scale memory demands. Moreover, workloads leveraging memory-intensive optimizations, such as KV caching and RAG, frequently initiate large-scale memory transactions, exacerbating memory bandwidth constraints. As a result, architectures optimized for computational throughput ironically become inadequate for supporting sustained, high-bandwidth memory traffic, leading to significant inefficiencies and performance degradation.

**Network communication.** Efficient network communication is indispensable as sophisticated parallelization strategies distribute workloads across multiple nodes. Conventional tightly integrated architectures predominantly optimize intra-node (and intra-rack) computational capabilities while overlooking extensive inter-node (and inter-rack) communication demands emerging at large scales. As parallelization expands across many nodes, the frequency and volume of inter-node communication grow substantially, often becoming several to tens of times larger than the actual GPU-resident data, surpassing available network bandwidth. For instance, as discussed previously, GPUs collectively manage intermediate activations, optimizer states, and gradient updates totaling hundreds of TBs per iteration in large-scale LLM training scenarios. The required inter-GPU communication arising from frequent synchronization of attention vectors, gradients, and KV caches can escalate to PB-scale data transfers per iteration, exceeding original GPU-resident data sizes and even at rack-scale. As a result, traditional architectures encounter severe network bottlenecks, impeding efficient GPU synchronization and effective scaling across large-scale data center deployments. Note that the immense communication overhead and challenges again underscore the critical importance of scalable, high-bandwidth, low-latency interconnect infrastructure.

**Latency sensitivity.** The latency sensitivity introduces critical challenges, particularly during inference phases such as decode operations in auto-regressive scenarios. Real-time AI workloads require low-latency responses, mandating exchanges of intermediate computational results among GPUs with precise synchronization

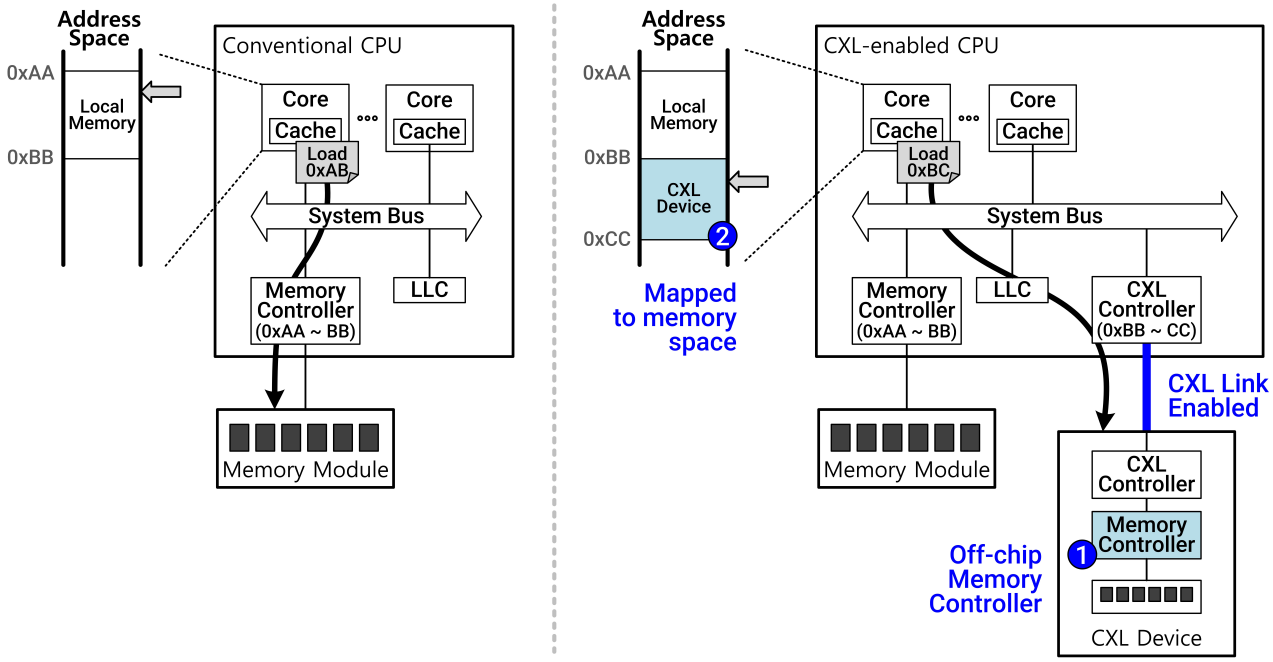



Figure 24: Relocation of memory controller in CXL.

as soon as possible. However, conventional infrastructures incur significant latency from frequent intermediate data movements across nodes and additional software overhead introduced by traditional communication stacks (e.g., Ethernet or InfiniBand). Specifically, network-based connection technologies involve substantial software overhead due to frequent privilege mode transitions of an operating system (e.g., kernel/user mode switches), redundant memory copy operations, interrupt handling, and protocol processing. These software-induced overheads typically increase latency by tens to hundreds of times compared to hardware-only operable interconnects such as CXL or NVLink, limiting achievable performance and scalability. This inherent latency restricts conventional architectures from meeting real-time responsiveness requirements, thereby constraining their applicability to latency-sensitive AI inferences.

These four performance dimensions (i.e., computational throughput, memory capacity and bandwidth, network communication, and latency sensitivity) involve competing constraints, making simultaneous optimization challenging (cf. Figure 22). Specifically, as shown in Figure 23, increasing computational throughput by extensive GPU parallelization intensifies demands on network bandwidth, exacerbating synchronization overhead. Likewise, enhancing memory capacity to accommodate massive parameter sets and activations introduces additional complexity and latency overhead due to frequent coherent data exchanges across nodes. Consequently, conventional architectures characterized by tightly integrated CPU-GPU modules lack the flexibility to scale compute, memory, and networking resources independently. This rigid coupling leads to suboptimal resource utilization and constrained overall system performance.

To address these architectural limitations, we believe that future AI data centers are required to adopt composable architectures enabling modular and independent scaling across compute, memory, and networking domains. CXL emerges as a practical solution, offering advanced features such as resource disaggregation, dynamic composability, and coherent memory pooling. By physically decoupling memory resources from processing units and facilitating direct cache-coherent access, CXL can also reduce latency, minimize synchronization overhead, and enhance overall system efficiency.

## 4.2. Background: Evolution of CXL and Composable Architectures

**Decoupling memory resources from CPUs: CXL 1.0.** To address the aforementioned scalability, flexibility, and performance issues, data center architectures can leverage various interconnect characteristics offered by CXL to facilitate architectural reconfiguration. In traditional computer architectures, memory controllers are tightly integrated within CPU packages, making memory expansion and physical disaggregation challenging. Physical separation of memory resources from CPUs is essential to accommodate dynamically changing work-

Table 1: Comparative analysis of different versions of CXL.

| **Feature** | **CXL 1.0** | **CXL 2.0** | **CXL 3.0** |
|---|---|---|---|
| Max Link Rate ($GTs$) | 32 | 32 | 64 |
| Flit 68-byte (up to 32 $GTs$) | ✓ | ✓ | ✓ |
| Flit 256-byte (up to 64 $GTs$) | – | – | ✓ |
| Memory Controller Decoupling | ✓ | ✓ | ✓ |
| Memory Expansion | ✓ | ✓ | ✓ |
| Memory Pooling | – | ✓ | ✓ |
| Memory Sharing | – | – | ✓ |
| Switching (Single-level) | – | ✓ | ✓ |
| Switching (Multi-level) | – | – | ✓ |
| Hierarchical-based Routing (HBR) | – | ✓ | ✓ |
| Port-based Routing (PBR) | – | – | ✓ |
| Hot-plug Support | – | ✓ | ✓ |
| Max Accelerator per Root Port | 1 | 1 | 256 |
| Max Memory Devices per Root Port | 1 | 256 | 4096 |
| Back-Invalidation | – | – | ✓ |
| Peer-to-Peer Communication | – | – | ✓ |
| Release Year | 2019 | 2020 | 2022-23 |

load demands, improve utilization of imbalanced memory capacities across nodes, and reduce TCO. However, due to architectural constraints, existing data centers have employed RDMA technologies to logically disaggregate memory, allowing multiple computing units to share these resources via software assistance [316–319]. Although RDMA methods provide architectural flexibility, their inherent software overhead, such as frequent privilege mode transitions, data serialization/deserialization, and redundant memory copy operations, results in significant performance degradation and substantial energy overhead from additional data movement and management.

CXL can directly address these fundamental limitations by physically and logically decoupling memory controllers from CPUs. As depicted in Figure 24, instead of embedding memory controllers within CPU packages, CXL relocates these controllers onto external memory modules, termed CXL *endpoints*, such as DRAM expansion cards or specialized memory devices. This critical architectural shift allows memory resources to scale independently beyond traditional CPU constraints, enabling true hardware-level memory disaggregation and composability. Unlike the conventional RDMA methods, CXL leverages the existing PCIe-based *physical layer* (PHY), but it delivers a direct, cache-coherent memory interface accessible via standard CPU load/store instructions. This design eliminates software-induced overhead, including context switches and redundant memory copy operations, by establishing direct hardware-mediated memory access paths. In addition, by implementing specialized logical layers (e.g., CXL link layer and transaction layer) atop the established PCIe infrastructure, CXL integrates into existing hardware ecosystems, simplifying architectural adoption without necessitating substantial hardware modifications.

Table 1 comprehensively compares the key features across CXL versions 1.0, 2.0, and 3.0, emphasizing the progressive enhancements in scalability, connectivity, and advanced functionalities. The initial specification, CXL 1.0 [59], introduced the key concept of memory controller decoupling, laying the foundational framework for composable infrastructure. However, practical scalability under CXL 1.0 remained limited as all types of CXL devices should be located within a node. Each CXL endpoint's external memory controller is constrained by a limited number of memory channels and physical factors, such as signal integrity and attenuation issues [59, 63]. Because of those, achievable memory capacities per CXL endpoint device typically remained within the range of 1–2 TB, complicating large-scale memory expansion. Given the limited number of endpoints that each node can accommodate, fully realizing extensive and scalable memory disaggregation required further architectural enhancements, motivating subsequent evolutions starting with CXL 2.0.

**Scalable and composable memory through switch-based architectures: CXL 2.0.** To overcome the memory capacity, endpoint scalability, and rigid connection limitations inherent to CXL 1.0, the CXL 2.0 specification [60] introduced an important architectural advancement: dedicated *switch*-based topologies. In contrast to the direct endpoint-to-host connectivity of CXL 1.0, which restricted scalability due to limited

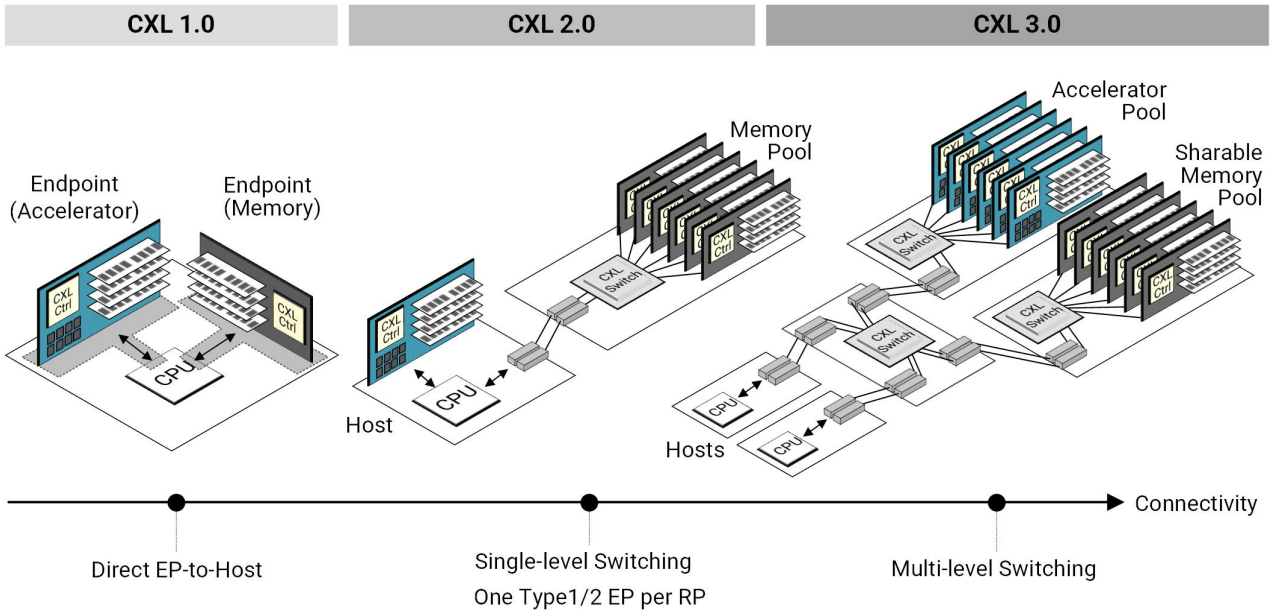



Figure 25: Evolution of CXL: from direct connections to multi-level switching.

memory channels and fixed endpoint configurations, CXL 2.0 enables flexible aggregation and management of multiple memory resources via intermediate CXL switches. As depicted in Figure 25, this introduction of an intermediate switching layer between compute nodes and memory endpoints allows individual hosts to access larger memory pools composed by multiple external endpoints, resolving the connectivity bottlenecks associated with earlier point-to-point CXL implementations.

Internally, CXL 2.0 switches can utilize high-bandwidth crossbar architectures, routing coherent memory transactions among numerous connected endpoints and compute nodes. This hardware-mediated coherent communication reduces latency in accessing large-scale external memory, eliminating substantial software-induced overhead observed in traditional RDMA-based network fabrics. Leveraging PCIe Gen5 technology (32 GT/s per lane), a CXL 2.0 switch can offer configurable multi-port interfaces, each capable of up to 64 GB/s bidirectional bandwidth in standard 16-lane configurations. As a result, a single CXL 2.0 switch can aggregate tens of TBs of memory per node, and because it can connect multiple nodes, it exceeds the scalability constraints inherent to endpoint-centric CXL 1.0 designs. This scalable topology supports modular system expansion and simplifies resource provisioning, decoupling memory expansion from rigid endpoint limitations. Advanced operational features introduced by CXL 2.0, including hot-plug support [60, 63], further enhance system flexibility by allowing dynamic addition or removal of memory endpoints with minimal operational disruption. In addition, host-specific static memory allocation features [63, 65, 66] enable memory resource management, empowering data centers to accommodate evolving workload requirements.

Despite these improvements, CXL 2.0 retained scalability constraints, particularly its inability to support hierarchical multi-level switch configurations. This restriction confined scalability to "single-layer switch" architectures, significantly limiting memory pool sizes and the number of devices per root port. Specifically, typical CPU architectures provide only a finite number of root ports, each constrained by fixed link capabilities and stringent bandwidth limits. Therefore, practical deployments of CXL 2.0 in practice support 4 to 16 memory expanders (i.e., Type 3 devices) per CPU root port, well below the theoretical maximum of 256 devices. This limitation becomes even more pronounced for accelerators (i.e., Type 1 and Type 2 devices), which require strict "one-to-one" mappings per root port to maintain cache coherence, restricting accelerator scalability and deployment flexibility.

Recognizing these scalability constraints underscored the necessity for further architectural advancements. This motivated subsequent technical enhancements introduced in CXL 3.0, including multi-level switch cascading, advanced routing mechanisms, and comprehensive system-wide memory coherence capabilities.

**True composability through multi-level switching and memory sharing: CXL 3.0.** Addressing the scalability and hierarchical limitations inherent to CXL 2.0, the CXL 3.0 specification [61] introduces crit-

ical architectural enhancements[3] that advance true composability and resource sharing in high-performance computing environments. Unlike CXL 2.0, which limited configurations to single-layer switch topologies and constrained the number of endpoint devices, CXL 3.0 explicitly supports multi-level switch topologies, termed *switch cascading*. This hierarchical fabric interconnects multiple CXL switches across several layers, overcoming single-layer limitations and increasing the number of endpoint devices (e.g., memory expanders and accelerators) that a CPU root port can coherently access.

Specifically, CXL 3.0 extends connectivity for memory expanders (Type 3 devices) per CPU root port to as many as 4,096 devices, facilitating extensive memory pools critical for large-scale AI and analytics workloads. Accelerator integration capabilities are similarly enhanced, supporting up to 256 accelerators (Type 1 and Type 2 devices) per root port, exceeding previous single-device constraints. These enhancements enable dynamic and flexible deployment of heterogeneous accelerator clusters within unified composable fabrics, thereby optimizing infrastructure efficiency to effectively accommodate evolving workload demands.

Internally, the multi-level switch fabric in CXL 3.0 introduces a novel *port-based routing* (PBR) mechanism, complementing the *hierarchical-based routing* (HBR) inherited from CXL 2.0. In contrast to HBR, which relies on fixed hierarchical paths and provides only static memory partitioning (exclusive allocations per host without dynamic sharing), PBR dynamically selects optimal routing paths based on real-time port conditions and network congestion. Importantly, PBR supports genuine multi-host memory sharing, enabling multiple hosts to concurrently and coherently access shared memory resources. This capability can improve traffic distribution, reduce latency, and mitigate communication bottlenecks, thereby overcoming limitations of the earlier static partitioning approaches inherent in CXL 2.0.

In particular, CXL 3.0 introduces robust multi-host memory sharing and comprehensive system-wide cache coherence capabilities, largely enabled by the advanced PBR mechanism. This architectural enhancement enhances computational efficiency, reduces latency, and lowers overhead in large-scale AI workloads. For example, accelerators and compute nodes can directly and coherently share essential data structures, such as embedding tables, KV caches, and intermediate activations, without redundant data transfers or complex software interventions. Furthermore, CXL 3.0 allows accelerator-local high-bandwidth memories to be unified into a single coherent memory pool, broadening their applicability across diverse system configurations. The practical implications and performance advantages of this genuine memory sharing approach are explored in detail in Section 5.2.

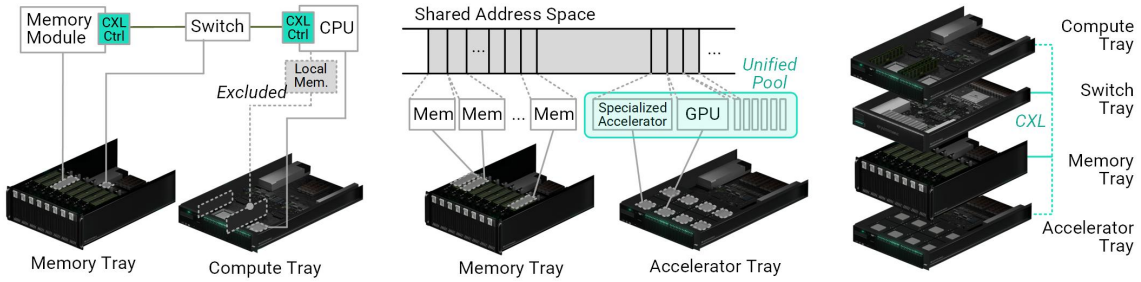
Although CXL 3.0 introduces extensive enhancements and maintains backward compatibility with CXL 2.0, its practical adoption requires specific hardware modifications across CPUs, switches, and endpoint devices, to fully leverage the new capabilities. Specifically, while the PBR mechanism primarily operates within CXL switches, corresponding hardware adaptations at endpoint devices are also necessary. Endpoints must handle larger flit sizes (256-byte flits in PBR mode compared to 68-byte flits in HBR mode). In addition, as CXL 3.0 facilitates genuine multi-host memory sharing with comprehensive cache coherence, memory expanders must implement advanced coherence mechanisms, such as back-invalidation, ensuring consistent data visibility and integrity across all shared resources. Endpoints may also support advanced CXL 3.0 features, including direct peer-to-peer communication, enabling accelerators to directly exchange data within a single host domain or access memory resources on other endpoints without host mediation. These features maximize the scalability and flexibility provided by the PBR-enabled fabric architecture, reducing communication latency and overhead.

## 4.3. CXL-Enabled Modular Tray and Rack Architecture for AI Data Centers

Building upon the composability and multi-host memory-sharing capabilities introduced by CXL 3.0, this subsection discusses how these architectural innovations enable a modular, tray-based design for modern AI data centers in practice. Unlike traditional tightly integrated accelerator nodes, *tray-based modular systems supported by CXL* allow independent scaling and dynamic reconfiguration of resources since CXL enables spatially separated placement of endpoints. Each CXL-enabled tray functions as a standardized hardware unit exclusively dedicated to a specific resource type, such as accelerators, CPUs, or memory, thereby enabling precise resource scaling, simplified maintenance, and agile adaptation to evolving AI workload demands.

**Modular, tray-based design enabled by CXL technology.** In CXL-based modular architectures, memory resources are fully decoupled from compute and accelerator modules, fundamentally changing resource management strategies in data centers. As illustrated in Figure 26a, *memory trays* exclusively integrate DRAM modules aggregated via dedicated CXL controllers or CXL switch(es), forming composable memory pools. These

[3]In this section, references to CXL 3.0 encompass all subsequent revisions within the CXL 3.x series, including versions such as 3.1, 3.2, and beyond.




(a) Disaggregated memory tray. (b) Disaggregated accelerator pool. (c) Connected trays.

Figure 26: Tray-based disaggregation with CXL cache-coherent sharing.

trays are not statically bound to specific compute nodes; instead, they can be flexibly allocated and shared among multiple accelerator or compute trays, enhancing scalability and operational flexibility. For instance, during intensive training phases of large-scale transformer models, additional memory trays can be provisioned to accelerator trays, accommodating increased memory demands without modifying CPU configurations.

Although the high-level concept of modular disaggregation has been previously proposed [300, 301, 303, 320], practical realization of true resource disaggregation was infeasible within traditional architectures. Historically, accelerators and memory modules functioned merely as passive peripherals incapable of independently managing memory coherence or direct access. As a result, CPUs hosted memory controllers in a tightly integrated manner and enforced cache consistency, coupling memory and accelerators within fixed server nodes. CXL resolves these constraints by relocating memory controllers externally and establishing standardized cache-coherent communication across all system components. Thus, accelerators can directly access and coherently share disaggregated memory resources without CPU mediation.

On the other hand, within this modular framework, dedicated *accelerator trays* integrate multiple GPUs or specialized accelerators interconnected via high-speed CXL interfaces. This configuration supports efficient cache-coherent communication and direct memory sharing among accelerators. As depicted in Figure 26b, employing standardized CXL interfaces enables the integration of various accelerators into a unified pool without coupling CPUs or memory devices. Moreover, the accelerators' local memory can be combined into a shared, coherent memory space, being able to reduce redundant data transfers across the accelerators. Such a design can enhance performance, particularly for frequently accessed intermediate data structures, including KV caches and intermediate activations. In parallel, *compute trays* exclusively host CPUs and, when necessary, network interface cards, deliberately excluding local memory to maintain strict resource disaggregation. Critically, dedicated *CXL switch trays* orchestrate coherent interactions among independently scalable compute, accelerator, and memory trays, facilitating dynamic resource allocation and composability in real time.

By dedicating each tray exclusively to a specific resource type, the full potential of CXL-based resource disaggregation is effectively realized. Accordingly, each resource, compute, accelerator, and memory, can independently scale in alignment with workload demands. This modular architecture significantly enhances operational efficiency and resource utilization, enabling rapid adaptation to diverse and evolving AI workloads, and providing a robust foundation for composable rack-level architectures discussed in subsequent sections.

**Composable rack architecture with CXL.** At the rack level, modular tray-based architectures enabled by CXL are organized to enhance functional clarity and resource utilization. Each rack comprises dedicated trays for networking, accelerators, CPUs, composable memory expanders, and storage. This design allows for specialized racks interconnected to form composable rows or flexible integration of various tray types within each rack, tailored to specific application needs.

Figure 26c illustrates one representative configuration of tray-based composability within a modular rack-level architecture enabled by CXL. In this particular layout, accelerator trays and compute trays are interconnected via centrally positioned memory trays managed by high-speed CXL switches. This configuration also provides coherent, low-latency access to pooled memory resources without the direct involvement of CPUs. Accelerator trays can therefore directly share and efficiently access these memory pools, significantly reducing redundant intra-rack data movements. Such a design is especially advantageous for workloads characterized by

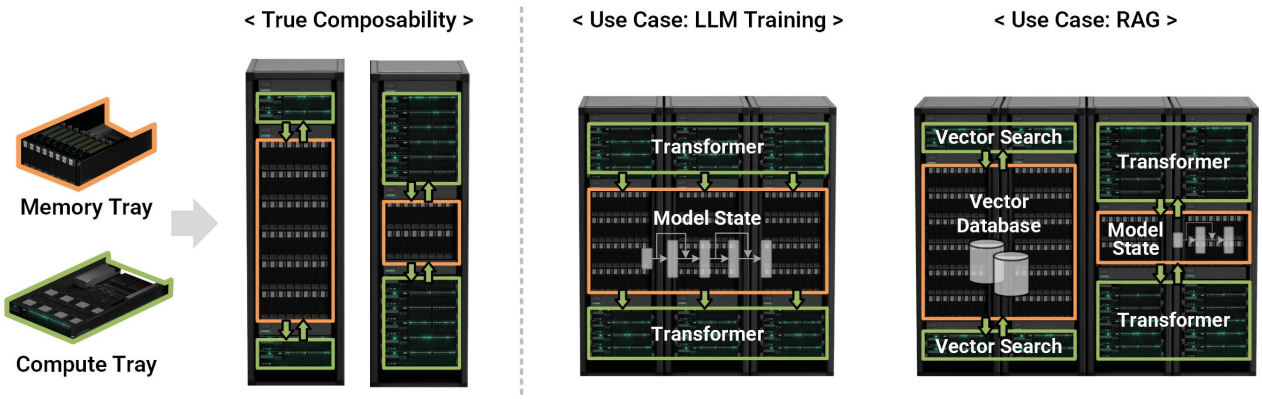



Figure 27: Composable rack architecture with CXL.

frequent reuse of intermediate data structures, including transformer inference scenarios and KV caching tasks.

In parallel, Figure 27 highlights another illustrative example of composable architectures deployed at rack scale, optimized specifically for diverse AI workloads. Here, accelerator trays and memory trays are fully modularized and decoupled, enabling dynamic scaling and flexible allocation of memory resources. Utilizing high-speed, cache-coherent communication through CXL, accelerators achieve efficient data sharing, thereby improving computational throughput and overall resource utilization. This modular approach particularly benefits workloads with dynamically varying computational and memory demands, such as large-scale LLM training and RAG workloads.

Note that a key advantage of the proposed modular architecture is importantly its optimized intra- and inter-rack networking strategy. Specifically, *this design integrates all racks and nodes within a row into a unified scale-up domain*. As discussed previously, traditional data center designs employing ToR switches inevitably classify inter-rack accelerator communication into the scale-out domain. When high-speed accelerator links such as UALink or NVLink are unavailable, unfortunately, intra-rack communication also depends on long-distance networks. This dependence significantly degrades performance, as each node is treated as an independent scale-out endpoint. Even when UALink or NVLink is available, non-accelerator devices within racks must still communicate through scale-out paths, exacerbating performance bottlenecks, particularly for workloads with intensive GPU-to-GPU interactions.

In contrast, leveraging CXL's support for hierarchical switch cascading, our modular architecture replaces conventional ToR switches by positioning dedicated CXL switch trays centrally as middle-of-rack (MoR) modules. Depending on bandwidth demands, additional CXL switch trays can be composably integrated, enabling flexible scaling and enhanced connectivity within racks and rows. Thus, racks and nodes within a row form an efficient scale-up domain over all interconnect fabrics rather than having long-distance networks. Traditional Ethernet or InfiniBand switches, primarily supporting inter-row communication, can be centrally organized into dedicated network racks. Through this hierarchical approach, network traffic is effectively aggregated and distributed, reducing congestion and latency, and maintaining balanced bandwidth allocation. Consequently, this infrastructure enables coherent, dynamic resource sharing across large-scale accelerator and memory pools, meeting the diverse requirements of modern AI workloads.

On the other hand, from an operational viewpoint, modular tray-based designs simplify maintenance and upgrades as well. Since memory, accelerator, compute, and networking components are physically and logically decoupled, each resource type can be independently replaced, upgraded, or scaled with minimal disruption. This modularity reduces system downtime, accelerates adoption of emerging technologies, and ensures infrastructure agility, enabling rapid adaptation to evolving workload demands while maintaining cost-effectiveness over time.

**How CXL meets diverse performance metrics.** Table 2 analyzes how the CXL-enabled tray-based architecture can address critical performance metrics required by modern AI infrastructures (cf. Section 4.1). Specifically, the proposed composable architecture provides a scalable and unified modular framework that enhances computational flexibility, expands memory capacity, optimizes memory bandwidth, reduces latency, and improves overall system scalability as below:

- **Scalability:** CXL's hierarchical switch design enables effective scalability through multi-level cascading of interconnect switches, facilitating incremental and seamless expansion of both computational and memory

Table 2: Performance comparison between conventional and CXL-enabled architectures.

| Essential Performance Metrics of Modern AI Infrastructure | Conventional Architecture | CXL-enabled Tray-based Architecture |
|---|---|---|
| Scalability | Node-level or rack-level scale-up with limited resource expansion | Row-level scale-up enabling flexible expansion of computational, interconnect, and memory resources |
| Latency | High-latency, protocol overhead and software intervention delays with RDMA (>1 $\mu$s) | Low-latency, hardware-mediated protocol and cache-coherent (100–250 ns) |
| Memory Capacity | Low and fixed, tightly coupled CPU and GPUs architecture (192–288 GB per GPU) | Massive and flexible, dynamic composable memory pool (> tens of TBs per node) |
| Memory Bandwidth | Low efficiency (memory copy for access external memory) | High efficiency (traffic reduction with coherent and pooled memory) |
| Computational Flexibility | Low flexibility (fixed or coarse-grained resource allocation) | High flexibility (dynamic and fine-grained resource allocation) |

resources without architectural bottlenecks. Specifically, this scalable architecture coherently aggregates thousands of accelerators and memory endpoints into unified resource pools, addressing critical performance metrics, including computational throughput, model size, and overall system scalability. Thus, CXL-based infrastructures can accommodate rapidly increasing model sizes and complex parallel workloads, eliminating the need for disruptive hardware upgrades and supporting continuous adaptation to evolving AI demands.

- **Latency:** CXL can fundamentally address latency constraints by providing direct, cache-coherent communication paths between memory and accelerator trays. Unlike conventional network-based approaches such as RDMA, which incur substantial protocol overhead and software-induced delays, CXL facilitates hardware-mediated memory interactions via direct load/store instructions. Specifically, accelerator and memory trays communicate through dedicated CXL interfaces, reducing round-trip latency. This low-latency communication is beneficial during latency-sensitive operations, including the decode phase of LLM inference. Thanks to the extended scale-up domain, CXL architectures can also optimize both network bandwidth utilization and latency sensitivity metrics, ensuring responsive and efficient AI deployments.
- **Memory capacity and bandwidth:** The modular tray-based memory configuration enabled by CXL aggregates numerous memory expansion modules, each providing substantial memory capacity and high-bandwidth interfaces. By coherently pooling these memory modules through dedicated CXL controllers or switches, accelerators can directly access shared memory resources, mitigating performance bottlenecks related to limited memory bandwidth and insufficient capacity. Specifically, such cache-coherent pooled memory structures, implemented via CXL.cache, remove redundant data transfers and memory traffic, greatly benefiting workloads characterized by frequent, high-bandwidth memory accesses, such as RAG and KV caching scenarios.
- **Computational flexibility:** CXL-based tray architectures support flexible and fine-grained resource scaling. Specifically, accelerator, memory, and compute trays can each be independently provisioned, upgraded, or reconfigured, enabling precise resource allocation aligned with specific workload requirements. For instance, GPU trays can scale to handle computationally intensive training phases or the inference prefill stage and reconfigure to meet stringent latency constraints during inference decode operations. This dynamic composability can address multiple critical performance metrics, including computational throughput, network bandwidth, and memory capacity, by optimizing resource allocation according to workload-specific demands, thus improving overall system utilization and operational efficiency.

By integrating these architectural solutions into a unified framework, CXL-based infrastructures effectively scale and dynamically adapt to evolving AI workload demands, optimizing key performance metrics. Specifically, extending the scale-up domain can eliminate communication overhead among resources, thus significantly enhancing overall performance. In addition, minimizing unnecessary scale-up switches further optimizes total cost of ownership. Moreover, cache-coherent memory sharing enables accelerators to directly access pooled memory without CPU intervention, reducing redundant data transfers and improving computational throughput as well as cost efficiency. Collectively, these capabilities position CXL-enabled modular architectures as a robust and flexible foundation capable of addressing the diverse and increasingly demanding requirements of large-scale AI workloads

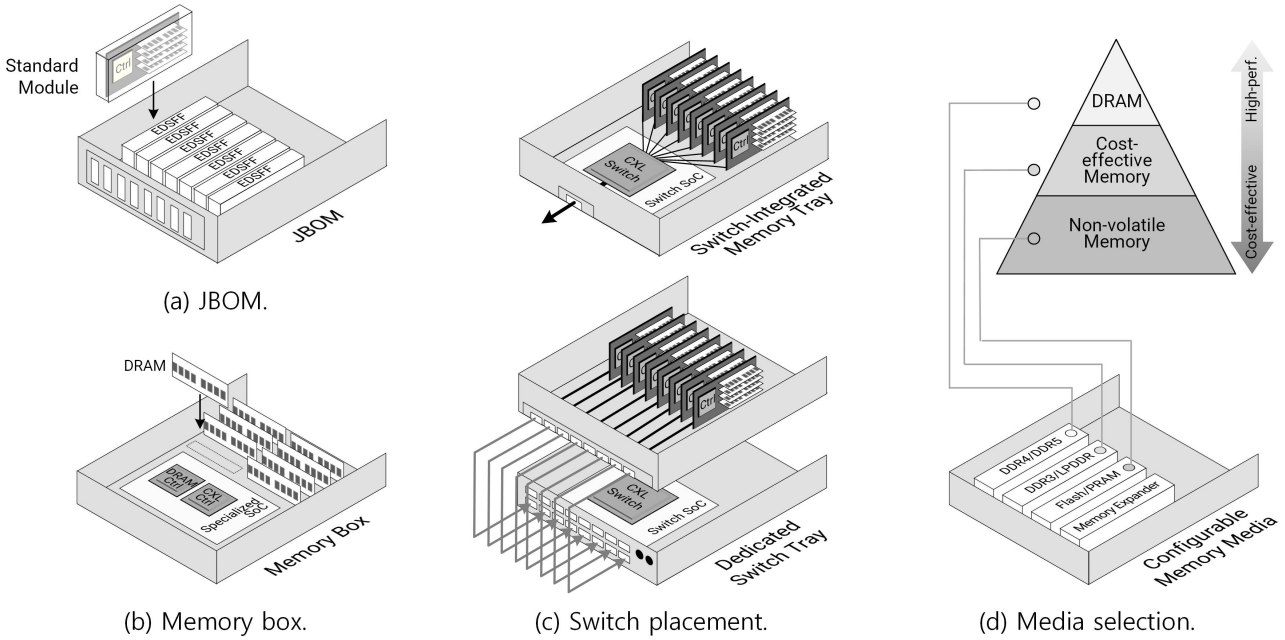



Figure 28: Tray-based disaggregation with CXL cache-coherent sharing.

# 5. Composable CXL Architectures: From Integration Strategies to Empirical Validations

In previous sections, we discussed how composable CXL architectures can address scalability and performance challenges inherent in modern AI infrastructures. Specifically, by supporting dynamic disaggregation and integration of memory and accelerator resources, these composable systems provide greater flexibility and adaptability compared to conventional RDMA-based approaches. However, the structural approach described thus far primarily focuses on modular tray-based configurations enabled by CXL. Practical implementations, such as the detailed composition of individual trays and the specific strategies for connecting GPUs or accelerators within these trays, require additional exploration. In this section, we specifically explore detailed implementation aspects, presenting integration strategies for composable CXL infrastructures. We also validate their effectiveness through empirical evaluations conducted across diverse real-world workloads.

## 5.1. Memory and Accelerator Management in Composable CXL Data Centers

This subsection discusses several considerations including architectural requirements for dedicated memory pools, efficient allocation and interconnection strategies for accelerators, and the development of comprehensive software frameworks necessary to manage memory and accelerator resources coherently.

**Dedicated memory pool implementation and management.** Implementing dedicated memory pools involves several architectural considerations. First, it is necessary to determine the hardware architecture and practical methods for memory tray implementation. Second, identifying cost-effective switch placement and defining their roles within the composable infrastructure is essential. Finally, selecting appropriate backend memory media for these dedicated memory pools must be thoroughly evaluated.

Initially, let us consider the hardware architecture and practical implementation methods for configuring memory trays. As illustrated in Figure 28a, memory trays can be configured either as *Just a Bunch of Memory* (JBOM) units or as specialized, dedicated memory boxes. In a JBOM configuration, memory expanders utilize standard enterprise and data-center storage form factors, such as EDSFF modules [321–324], arranged in arrays. Although many memory vendors adopt this standardized approach, it introduces increased costs and complexity due to vendor-specific performance variations and greater operational overhead. Specifically, replacing memory media in JBOM units requires simultaneous replacement of both CXL and memory controllers, despite their longer operational lifespans compared to memory media, thereby increasing the total cost of ownership.

Alternatively, memory trays can adopt dedicated memory boxes integrating specialized system-on-chips (SoCs) equipped with multiple DRAM and CXL controllers. As illustrated in Figure 28b, this configuration directly supports high-performance raw or *dual inline memory modules* (DIMM [325–327]) by fully decoupling memory controllers from backend memory media. Such decoupling reduces maintenance complexity and operational costs. Moreover, by embedding both CXL and memory controllers within the memory tray, data centers can reuse legacy DIMMs or older DDR memory modules already present, offering additional cost advantages. This design provides significant flexibility, enabling operators to customize tray specifications, including port counts, bandwidth, and memory module types, to align with specific performance targets and workload requirements. Direct control over memory modules also allows operators to optimize memory media selection (e.g., DDR3 [328–331], DDR4 [332–334], LPDDR [335–337]), balancing performance and cost efficiency from the data centers' viewpoint. However, this strategy increases complexity in data-integrity management, coordination of heterogeneous hardware components, and handling sophisticated SoC designs at some extents.

Another important architectural consideration involves determining optimal switch placement for interconnecting memory expanders or SoC-based controllers within memory trays. As illustrated in Figure 28c, integrating switches directly within each memory tray addresses compatibility concerns, version discrepancies, and functional variations among different expanders. Specifically, by treating each tray as a self-contained box, external systems can seamlessly abstract internal hardware variations and diverse memory technologies through standardized interfaces or controlled performance guarantees. This design aligns naturally with the previously discussed JBOM and dedicated memory-box architectures; however, similar drawbacks, particularly higher costs and limited flexibility, arise due to tight integration and reduced hardware adaptability. Alternatively, placing switches externally at dedicated switch trays or MoR/ToR positions allows memory trays to function passively, reducing cost, enhancing operational flexibility, and facilitating diverse, adaptive configurations tailored to specific data-center deployment scenarios.

In addition to tray configurations and switch placements, overall efficiency can be improved by diversifying memory media types and organizing them hierarchically within trays (cf. Figure 28d). Traditional DRAM modules (e.g., DDR4 or DDR5) are costly and offer limited flexibility for workload-specific tuning. Thus, employing cost-effective or power-efficient memory solutions, such as LPDDR or DDR3, can be beneficial. Moreover, integrating HBM modules as intermediate buffering layers within memory expanders or trays can enhance performance. Specifically, these HBM modules can accommodate variations in expander performance and mitigate latency introduced by integrated or external switches when constructing dedicated memory pools. Depending on bandwidth requirements and positioning within the system, reusing legacy or lower-cost HBM modules, such as older HBM versions or modules with fewer channels, can effectively balance performance requirements and reduce overall system costs. In addition, emerging non-volatile memory technologies, including flash memory or phase-change memory (PRAM), can provide data persistence capabilities where necessary, further optimizing performance and cost profiles.

**Accelerator resource management: Topologies and interconnect strategies.** Implementing accelerator composability requires careful consideration of several factors beyond basic interconnectivity. Specifically, topology design should reflect the unique characteristics of different accelerator vendors and consider various accelerator communication patterns and application-specific requirements. Here, we discuss accelerator resource management considerations in the context of LLM workloads exhibiting strong data locality and intensive data exchanges among adjacent accelerators, employing tensor parallelism. For general-purpose AI data centers characterized by random uniform traffic patterns, hybrid interconnect solutions will be addressed in Section 6.

In contrast to accelerator-optimized interconnect technologies that utilize a single dimensional topology (e.g., UALink and NVLink), CXL supports diverse topological configurations, providing flexibility for accelerator connectivity. Figure 29 compares the key characteristics of Clos [248, 338, 339], 3D-Torus [340–343], and DragonFly [344–346] topologies, which are widely employed interconnect techniques in modern data centers. Specifically, Clos networks deliver uniform bandwidth across nodes through multi-stage switch hierarchies, offering flexibility at the expense of increased complexity and higher implementation costs. In contrast, the 3D-Torus topology directly interconnects nodes in a three-dimensional mesh structure, supporting short-range communications at relatively lower costs. However, it can introduce bottlenecks under heavy long-range communication patterns. The DragonFly topology integrates fully connected local node groups with indirect inter-group links, balancing cost and performance, though certain traffic patterns may degrade overall efficiency.

Considering the communication characteristics of LLM workloads (i.e., intensive data exchanges among nearby accelerators), 3D-Torus or DragonFly topologies might initially appear attractive. However, these approaches become impractical due to the exponential growth in required switch counts as the number of accelerators scales. Thus, maintaining cache coherence among accelerators using a single-hop Clos topology emerges as

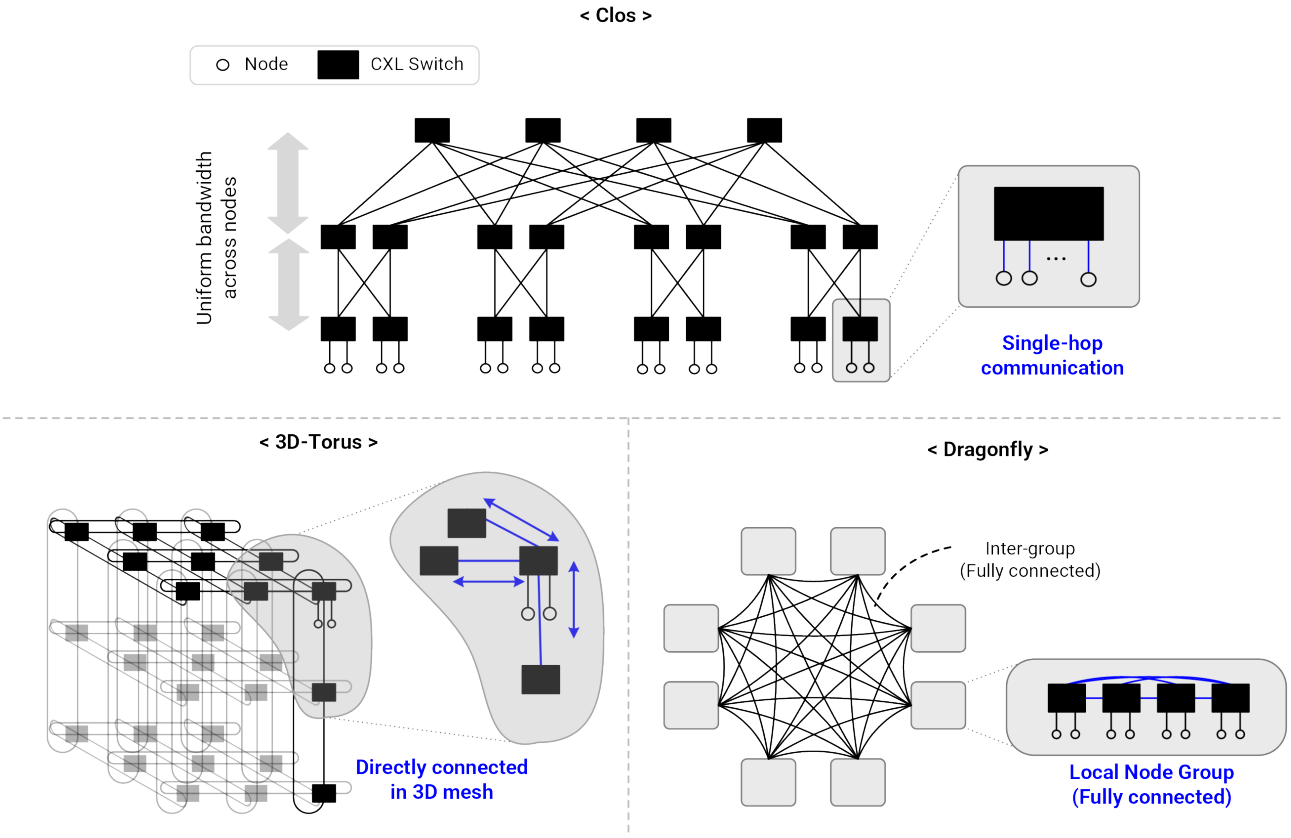
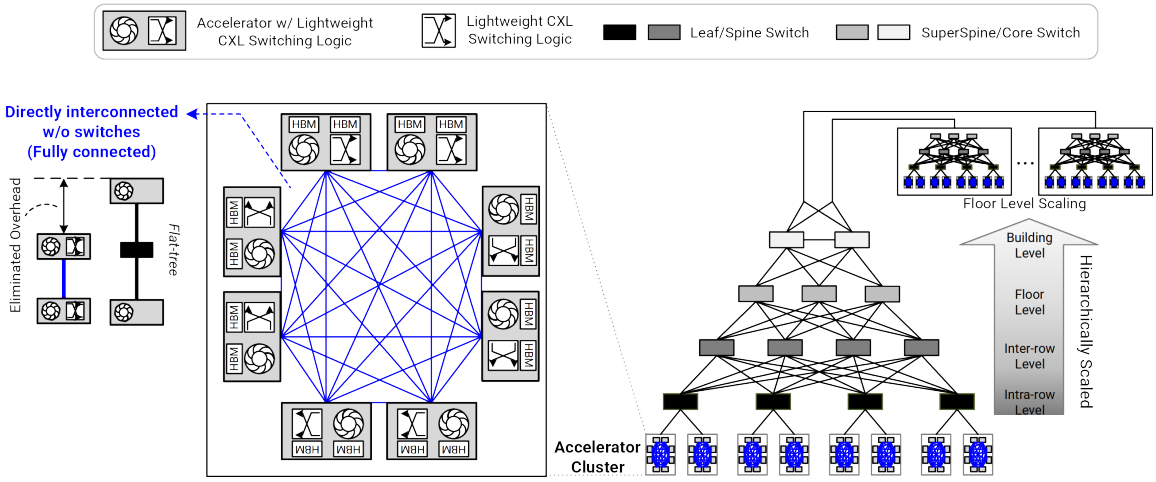



Figure 29: Topology comparison: Clos, 3D Torus, and Dragonfly.

the most practical solution. This design principle aligns closely with fundamental interconnect strategies utilized by NVLink and UALink (detailed in subsequent sections), enabling local accelerator communication at reasonable hardware cost. Nevertheless, the current CXL specification limits cache-coherent accelerator connections to 256, necessitating alternative topological strategies or the incorporation of additional switches to support larger-scale deployments.

A more aggressive integration approach is illustrated in Figure 30a, where accelerators are interconnected using a fully-connected topology without intermediate switches. In this configuration, each accelerator integrates simplified and lightweight internal CXL switching logic, eliminating the overhead associated with external single-hop Clos topologies and additional interconnect hardware within clusters. This fully-connected architecture optimizes data transfers among adjacent accelerators, making it advantageous for LLM workloads exhibiting intensive localized communication patterns. Accelerator clusters structured in this manner can scale hierarchically through external CXL switches across multiple physical levels, such as racks and floors, as depicted by the hierarchical fully-connected topology across multiple physical levels in Figure 30b. Furthermore, leveraging accelerator-local HBM enhances intra-cluster communication efficiency. While this fully-connected approach can reduce accelerator-to-accelerator data exchange overhead, it demands substantial design revisions and implementation complexity for both accelerators and Type 1 and Type 2 CXL controllers. In addition, inter-cluster communication through external switches may introduce performance imbalances and requires careful management of accelerator-generated bandwidth traffic. Therefore, comprehensive evaluation and meticulous design are important for practical deployment at scale.

**Unified management frameworks for composable resources.** The practical deployment of composable CXL infrastructures in AI data centers may require careful consideration of software-related architectural issues as well. Traditional static memory allocation incurs excessive data movements and latency, for frequently accessed data structures such as embedding tables in recommendation models and attention caches in transformer inference. To address these limitations, advanced software frameworks can be employed. These frameworks are required to incorporate predictive memory placement, proactive cache warming strategies, and adaptive eviction policies based on real-time access patterns. For example, dynamically allocating frequently accessed attention




(a) Directly interconnected accelerator cluster.

(b) Hierarchically scaled accelerator clusters.

Figure 30: Switch placement options.

caches closer to accelerators can reduce latency and enhance throughput in LLM inference scenarios utilizing tensor parallelism.

Effective accelerator resource management also impacts overall system performance. As discussed previously, conventional GPU-CPU architectures tightly couple computational and memory resources, often leading to inefficient accelerator utilization under varying workloads. In contrast, composable architectures support independent scaling and dynamic resource allocation tailored to evolving workload demands. Achieving this flexibility may necessitate sophisticated software frameworks capable of real-time workload monitoring, predictive resource allocation, priority-driven scheduling, and rapid reconfiguration. Such frameworks are also required precisely coordinate hardware arbitration and minimize inter-accelerator communication latency.

In addition, integrated management of memory and accelerator resources may demand centralized monitoring frameworks beyond traditional static approaches. For example, static resource management techniques are inadequate for handling dynamic resource contention in composable systems. Advanced software solutions can be therefore needed to incorporate real-time telemetry collection, comprehensive performance analytics, including coherent memory efficiency, resource allocation latency, and contention metrics, and automated corrective actions. Future software frameworks could further incorporate reinforcement learning-based orchestration and predictive monitoring optimized explicitly for CXL environments, proactively resolving resource contention and dynamically adjusting allocations. Such advanced capabilities would significantly enhance responsiveness and efficiency for latency-sensitive AI workloads.

## 5.2. Practical Case Studies of CXL Infrastructure in AI Workloads

Building on the theoretical advantages of the previously introduced CXL infrastructure, this subsection provides empirical evidence and practical insights into the advantages of composable CXL infrastructures through real-system prototype evaluations. Specifically, we evaluate representative contemporary workloads from AI and HPC domains, including RAG, graph-based RAG (Graph-RAG), deep learning recommendation models (DLRM), and MPI-based scientific computing applications. Our prototype demonstrates that composable CXL infrastructure mitigates critical performance bottlenecks such as excessive latency, significant data movement overhead, and inefficient memory utilization, which are prevalent in traditional architectures dependent on RDMA-based networking.

Figure 31 summarizes the performance improvements achieved by CXL compared to conventional systems across each scenario. Specifically, AI-based search workloads employing RAG and Graph-RAG integrated with LLMs exhibit a 14.35× reduction in execution time, alongside up to 21.1× decreases in data movement overhead relative to conventional architectures. Similarly, embedding-intensive DLRM workloads demonstrate approximately 3.32× faster inference execution and 2.71× accelerated tensor initialization. Furthermore, MPI-based HPC applications benefit from CXL-enabled direct memory sharing, achieving execution time improvements of

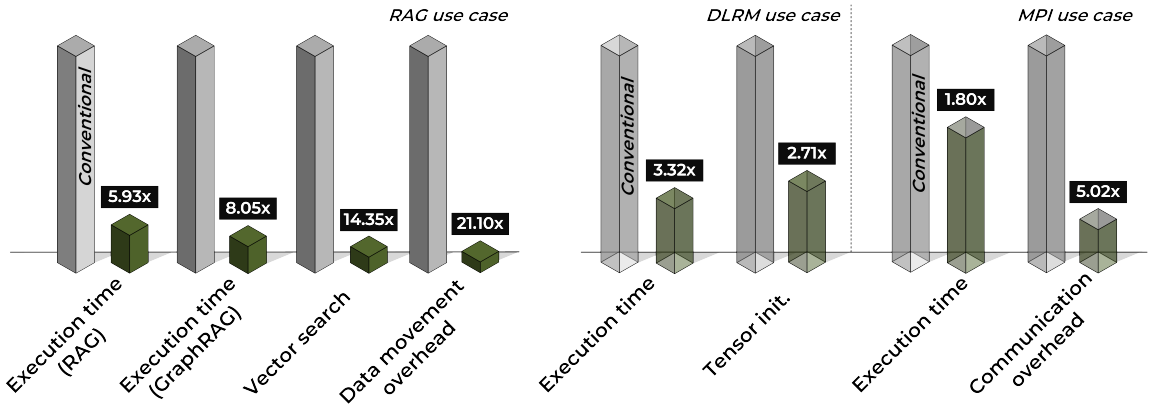



Figure 31: Summary of performance gains: RAG, Graph-RAG, DLRM, and MPI.

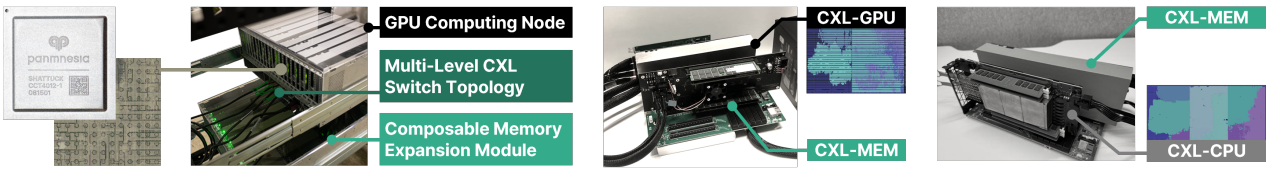
approximately 1.8× and reducing communication overhead by up to 5.02×.

In the subsequent subsections, we detail these scenarios, illustrating how the proposed CXL prototype addresses the performance challenges for each workload. Building upon these empirical insights, we also outline critical architectural implications, providing several guidelines for integrating CXL into future AI data center designs.

**Experimental infrastructure.** To evaluate composable CXL-based architectures, we developed a unified experimental infrastructure employing a real-system prototype compliant with the CXL 3.0 specification. Figures 32a and 32b illustrate our silicon-proven CXL controller IPs and a practical setup, respectively. This setup consists of GPU computing nodes interconnected with composable memory expansion modules through a hierarchical CXL switch topology. The memory expanders and switches leverage standard CXL hardware stacks, while the GPU and CPU nodes integrate customized CXL hardware directly within their root ports and endpoint complexes. Due to the current absence of commercially available GPUs and CPUs supporting CXL 3.0, we utilized open-source Vortex GPU [347–350] and RISC-V CPU [351, 352] microarchitectures, which we modified to incorporate essential CXL functionalities of our controller IPs. Prototype implementations of these modified GPUs and CPUs are depicted in Figures 32c and 32d, respectively.

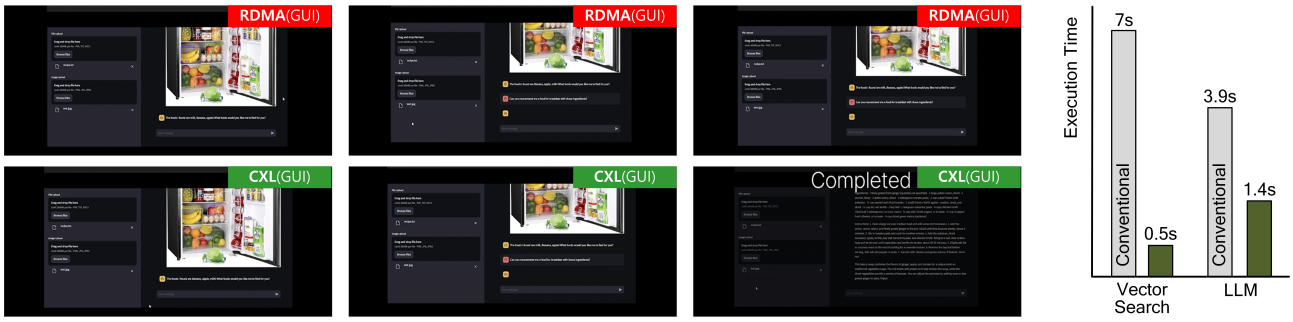
Memory modules are organized into coherent, dynamically composable pools, presented to GPU computing nodes as distinct non-uniform memory access (NUMA [353–355]) domains. This composable design enables GPU nodes to directly access shared memory resources, bypassing traditional CPU-mediated memory management or RDMA-based communication protocols. Although our evaluations utilized lightweight, open-source CPU and GPU implementations, our silicon-proven CXL controller and hardware stack IPs can be integrated with various third-party accelerators, NPUs, GPUs, and memory expanders. Specifically, these IPs can be modified to accommodate diverse cache and system-bus interfaces, facilitating straightforward integration into existing hardware platforms.

**RAG use case: Accelerating interactive retrieval and inference workloads.** Interactive retrieval tasks involving vector matching and real-time inference present significant challenges for traditional infrastructures due to high latency and intensive memory demands. To demonstrate the practical advantages of composable




(a) CXL IPs. (b) Real-system Prototype. (c) CXL GPU. (d) CXL CPU.

Figure 32: A experimental end-to-end infrastructure compliant with the CXL 3.0 specification.



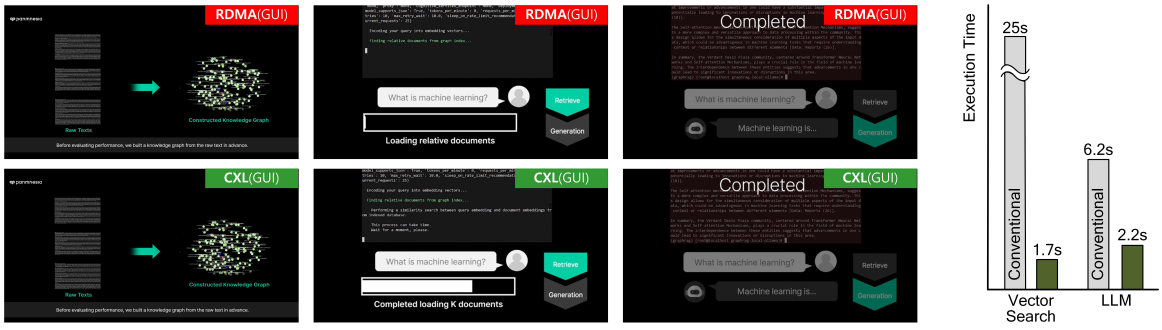

(a) Beginning ($t_0$). (b) Middle ($t_1$). (c) End ($t_2$). (d) Execution time.

Figure 33: RAG use case: recipe recommendation (*Demo Video: [Link]*).

CXL infrastructures, we evaluated a user-friendly RAG scenario integrated with a contemporary LLM. As depicted in Figure 33, this scenario represents a recipe recommendation system where users upload images of available food ingredients (typical refrigerator items) and specify preferred meal categories, such as breakfast or dinner. Embedding vectors for user-uploaded images are generated using a pre-trained visual-language model (e.g., CLIP [356–358]), ensuring accurate semantic representations. Subsequently, these embeddings are matched against pre-existing recipe embeddings stored in the system. Compared to conventional infrastructures utilizing RDMA-based interconnects, the composable CXL architecture demonstrates notable performance improvements in vector retrieval. These enhancements arise from reduced memory-access latency and decreased software overhead, which occur in RDMA-based systems.

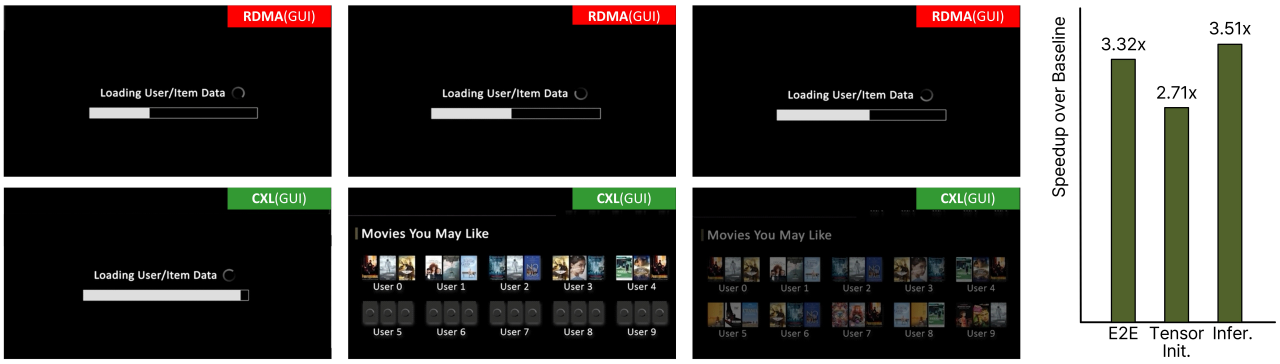
Following vector retrieval, the retrieved embeddings directly served as inputs for the LLM-based inference phase, generating contextually relevant recipe recommendations. While the conventional RDMA-based systems typically exhibit retrieval and inference latencies ranging from several hundred milliseconds to seconds, our composable CXL infrastructure achieved lower latencies, enabling responses within tens of milliseconds. As illustrated in Figure 33d, quantitative evaluations demonstrated that the composable CXL infrastructure completed the vector search and LLM in 0.5s and 1.4s, respectively, which are 14× and 2.78× faster than the baseline system. Such latency reductions are critical for user-facing applications, such as recommendation systems, where rapid, interactive responses enhance user experience and satisfaction.

**Graph-RAG use case: Accelerating knowledge graph-based retrieval and inference.** Graph-based RAG workloads, which integrate structured knowledge retrieval with inference, often experience substantial performance degradation due to high latency when accessing external memory resources. To address this challenge, we evaluated a composable CXL infrastructure using a Graph-RAG application scenario that integrates structured knowledge retrieval with LLM inference [359–362]. As depicted in Figure 34, the evaluation involved




(a) Beginning ($t_0$). (b) Middle ($t_1$). (c) End ($t_2$). (d) Execution time.

Figure 34: Graph-RAG use case: knowledge graph and query retrieval.




(a) Beginning ($t_0$). (b) Middle ($t_1$). (c) End ($t_2$). (d) Execution time.

Figure 35: DLRM use case (*Demo Video: [Link]*).

two primary operational phases: knowledge graph construction, followed by query-driven retrieval and inference.

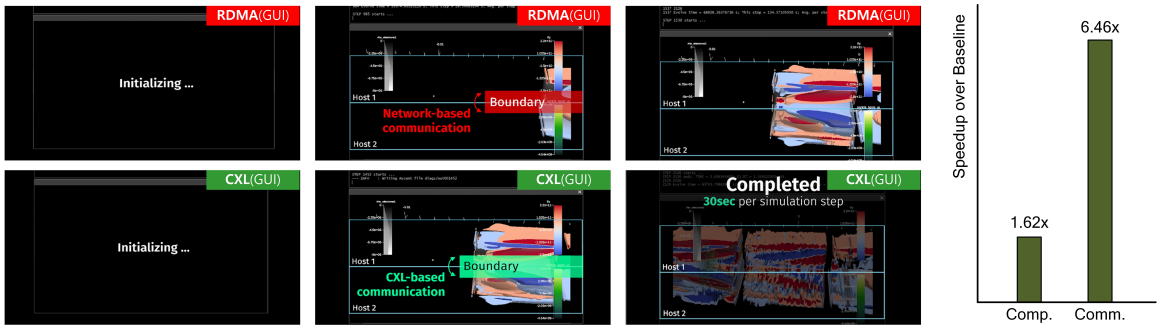
Initially, raw textual data sources were processed using standard graph embedding techniques (e.g., RDF embeddings [363–366] or graph neural networks [367–369]) to construct structured knowledge graphs optimized for efficient semantic retrieval. Subsequently, user queries were transformed into embedding vectors and matched against the structured knowledge graph using approximate nearest neighbor search methods such as HNSW [370–372] or FAISS [373, 374], enhancing retrieval speed. The retrieved embeddings then served as contextual inputs for the LLM inference process, generating coherent and contextually accurate responses.

Compared to the composable CXL infrastructure, the conventional baseline system utilized RDMA over InfiniBand, incurring storage and software-induced latencies during vector retrieval. Empirical analysis demonstrated that the composable CXL architecture reduced total workflow execution time by approximately 8.05× relative to the conventional RDMA-based baseline. In particular, while the conventional system takes tens of seconds, the composable CXL architecture completes the vector search and LLM inference phases in only 1.7s and 2.2s, respectively (cf. Figure 34d). This latency reduction and execution speed improvement resulted from the cache-coherent memory pools enabled by CXL, eliminating redundant data copying, bypassing software overhead, and providing direct hardware-mediated memory access.

**DLRM use case: Accelerating deep learning recommendation workloads.** DLRM workloads [375–378] require efficient embedding lookups, posing challenges related to memory capacity and latency for traditional data center infrastructures. Given these substantial memory and computational demands, it is essential to evaluate composable CXL-based architectures capable of addressing these issues. To this end, we analyzed embedding-intensive tensor initialization and inference phases representative of realistic recommendation scenarios, utilizing the composable infrastructure described previously.

As depicted in Figure 35, the evaluation utilized embedding tables containing hundreds of GBs of parameters, reflecting large-scale production recommendation systems. During tensor initialization, embedding tables were loaded into memory, which is a phase in which the composable CXL infrastructure demonstrated notable performance improvements compared to the conventional RDMA-based baseline. Specifically, the baseline system employed RDMA, commonly deployed in production environments and characterized by higher software-induced communication overhead and latency. In contrast, the composable infrastructure reduced initialization latency and communication overhead via direct hardware-mediated memory access and coherent memory pooling. Following tensor initialization, both infrastructures executed repeated inference computations, simulating realistic operational conditions and verifying sustained performance over multiple inference cycles. Due to accelerated tensor initialization, the composable CXL infrastructure transitioned more rapidly into inference execution, improving overall responsiveness compared to the baseline's prolonged initialization delays.

As shown in Figure 35d, empirical evaluations demonstrate that the composable CXL infrastructure achieves an overall throughput improvement of approximately 3.32× compared to the RDMA-based system. As described previously, the composable architecture accelerates tensor initialization and inference phases by 2.71× and 3.51×, respectively. This performance improvement resulted from CXL's cache-coherent memory pools, enabling accelerator nodes direct hardware-level memory access without network-based communication stack overhead, reducing latency and data transfer overhead. Practically, such performance enhancements translate




(a) Beginning ($t_0$). (b) Middle ($t_1$). (c) End ($t_2$). (d) Execution time.

Figure 36: MPI use cases: plasma simulation (*Demo Video: [Link]*).

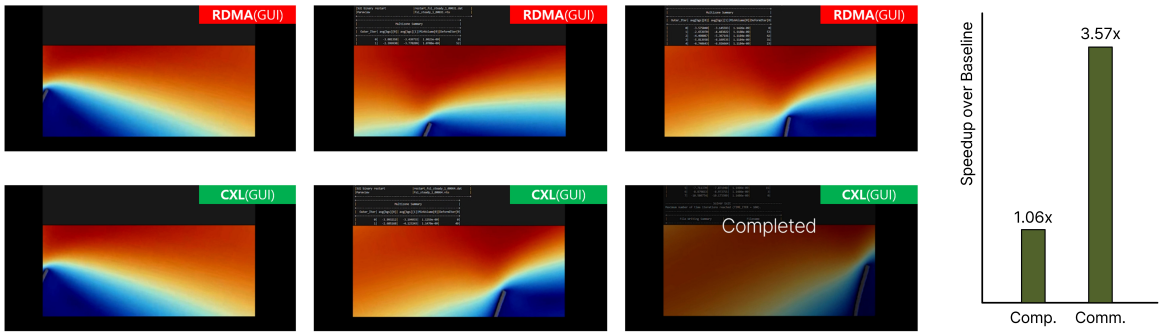
into improved user experiences for large-scale commercial platforms, including personalized content delivery in e-commerce and streaming services. Consequently, adopting composable, independently scalable CXL infrastructures enhances data center efficiency, meeting the evolving requirements of recommendation workloads.

**MPI-based scientific applications: Evaluating memory sharing with CXL.** MPI-based scientific computing applications experience inter-node communication overhead and synchronization latency, limiting performance scalability in traditional network-based architectures [379–382]. Although our primary focus remains on AI workloads, we evaluated representative MPI-based scientific computing applications to demonstrate the advantages of direct memory sharing enabled by composable CXL infrastructures. MPI-based scientific simulations, such as particle-in-cell (PIC) plasma simulations [383–385] and computational fluid dynamics (CFD) simulations [386, 387], involve intensive inter-node data exchanges and frequent synchronization of simulation states, closely resembling communication patterns prevalent in distributed AI workloads. Typically, these MPI applications partition computational domains across multiple nodes and regularly synchronize boundary conditions and intermediate simulation data.

To evaluate how composable CXL infrastructure mitigates these communication bottlenecks, we implemented two representative MPI scenarios. Figure 36 shows the first scenario utilizing WarpX [383], a PIC framework, simulating interactions among hundreds of millions of charged particles (e.g., electrons and protons) distributed across multiple computational nodes. The conventional RDMA-based infrastructure relying on InfiniBand incurs significant overhead due to system software involvements and data copies across different device/network domains. In contrast, our composable CXL-based setup enabled host CPUs to directly store boundary-related particle data into dynamically composable memory regions shared by CXL.cache. Other nodes accessed this data immediately through direct load operations without invoking traditional software-driven network protocols, significantly reducing communication overhead and latency. Quantitatively, as depicted in Figure 36d, the CXL-based configuration eliminates explicit synchronization overhead inherent to conventional RDMA-based implementations. This elimination results in reductions of computation and communication latencies by 1.62× and 6.46×, respectively.

In the second scenario, illustrated in Figure 37, we evaluated a CFD simulation involving intensive synchronization of fluid states across domain partitions. Traditional RDMA-based network communications incurred substantial delays and overhead during synchronization events. By adopting the composable CXL infrastructure, host CPUs directly accessed fluid simulation states stored in composable memory pools, replacing conventional RDMA-based communication with direct shared-memory interactions. In this approach, individual hosts independently performed computations required for CFD simulation. Although certain calculations spanned multiple CPUs, collective communication or explicit synchronization was unnecessary for data aggregation or updates. This is because data consistency and coherence are managed through CXL.cache, enabling CPUs to access uniform memory spaces as if accessing local memory. Consequently, synchronization overhead was significantly reduced, achieving approximately a 1.06× reduction in computation time and a 3.57× reduction in communication time compared to the conventional RDMA-based baseline (cf. Figure 37d).

Performance analyses highlight two principal advantages of composable CXL infrastructure. First, the elimination of explicit RDMA-based network operations reduces latency and software overhead through direct, cache-coherent, hardware-mediated memory sharing. Second, resource disaggregation and memory pooling sig-




(a) Beginning ($t_0$). (b) Middle ($t_1$). (c) End ($t_2$). (d) Execution time.

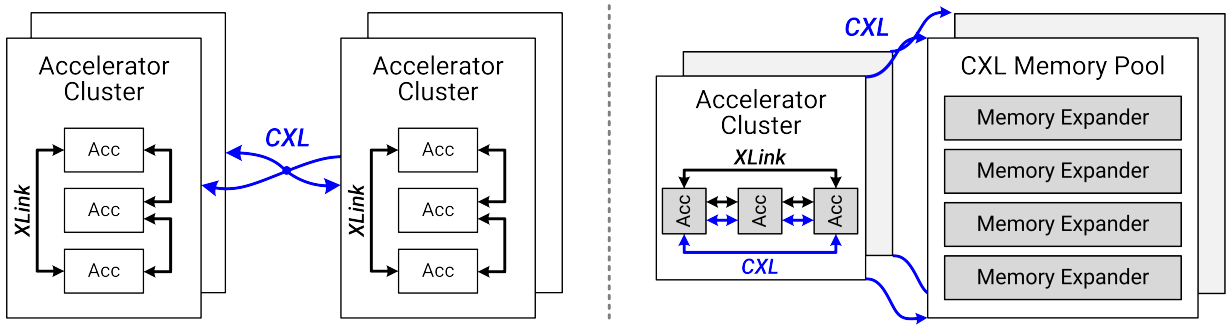
Figure 37: MPI use cases: fluid simulation.

nificantly streamline data management, improving scalability and operational efficiency in distributed computing environments. Practically, these enhancements substantially increase scalability and efficiency of large-scale scientific research infrastructures, particularly in domains such as climate modeling, astrophysics, and fusion research, where simulation speed directly impacts research productivity and accuracy. Although scientific MPI workloads fundamentally differ from AI-specific applications, we believe that these evaluations offer valuable insights into potential advantages of employing similar memory-sharing strategies within large-scale distributed AI infrastructures.

# 6. Beyond CXL: Optimizing AI Resource Connectivity via Hybrid Link Architectures

While CXL addresses critical memory-capacity expansion and coherent data-sharing challenges, integrating complementary interconnect technologies enables targeted enhancements for specific accelerator-centric workloads requiring efficient intra-accelerator communication, being able to support diverse workload demands and optimizing overall data center efficiency.

Two prominent accelerator-focused interconnect technologies are *Ultra Accelerator Link* (UALink) and NVIDIA's *NVLink*, collectively termed *Accelerator-Centric Interconnect Link* (XLink) in this technical report. XLink technologies provide direct, point-to-point connections explicitly optimized for accelerator-to-accelerator data exchanges, enhancing performance within tightly integrated accelerator clusters. In contrast to CXL, these XLink technologies do not support protocol-level cache coherence or memory pooling; instead, their focus is efficient, low-latency data transfers among accelerators with a single-hop Clos topology interconnect architecture. While both UALink and NVLink share this common objective, they differ in implementation specifics: UALink




(a) Accelerator-centric clusters. (b) Tiered memory architectures.

Figure 38: A high-level viewpoint of hybrid link architectures (CXL-over-XLink).

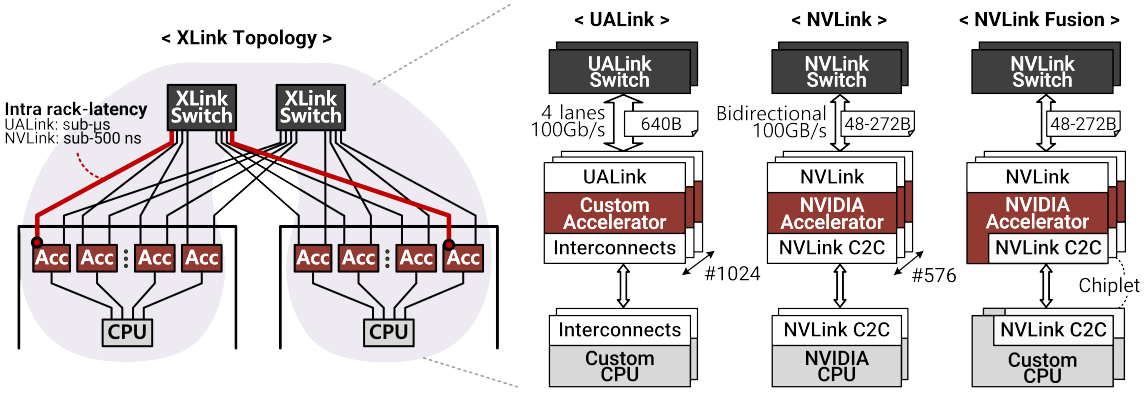



Figure 39: Accelerator-centric interconnects.

employs Ethernet-based communication optimized primarily for large-sized data transfers, whereas NVLink utilizes NVIDIA's proprietary electrical signaling, tailored for small-to-medium-sized data exchanges, such as tensor transfers and gradient synchronization between GPUs.

Integrating CXL and XLink into a unified data center architecture, termed *CXL over XLink*, including *CXL over NVLink* and *CXL over UALink*, leverages their complementary strengths to optimize overall system performance. As depicted in Figures 38a and 38b, this integration adopts two architectural proposals: i) "accelerator-centric clusters," optimized specifically for rapid intra-cluster accelerator communication, and ii) "tiered memory architectures," employing disaggregated memory pools to handle large-scale data. XLink typically supports optimized intra-node communication using direct, single-hop Clos topologies, making it effective for bandwidth-sensitive operations such as frequent tensor exchanges and gradient synchronization. However, the single-hop Clos topologies limit scalability, restricting the maximum number of connected accelerators and memory devices. In contrast, CXL enables scalable accelerator interconnections through multi-level switch cascading, facilitating diverse topologies and coherent memory pooling across multiple clusters or data centers. This allows dynamic memory allocation critical for memory-intensive workloads such as KV caching and RAG. Furthermore, CXL supports efficient inter-node data sharing through protocol-level cache coherence and instruction-level memory transactions, minimizing redundant data transfers and improving memory utilization. Composable disaggregation physically separates computational resources from memory pools, enabling independent scaling, simplified maintenance, and flexible hardware upgrades. Compute nodes interconnected via XLink, alongside memory resources managed through CXL, improve operational flexibility, accommodating rapid transitions between compute-intensive training and latency-sensitive inference workloads. In addition, memory resources can be organized hierarchically, optimizing allocation strategies according to specific performance and capacity requirements.

In this section, we first provide an overview of XLink technologies, emphasizing key architectural features and optimizations of UALink and NVLink. We then discuss several hybrid architecture strategies of CXL-over-XLink, highlighting how the complementary strengths of CXL and XLink address diverse and evolving workload requirements in modern AI data centers.

## 6.1. Background on Accelerator-Centric Interconnects: UALink and NVLink

**Ultra Accelerator Link.** UALink is an accelerator-centric interconnect optimized for accelerator-to-accelerator communication within data centers [76, 77]. In contrast to CXL, which emphasizes memory disaggregation, coherent memory management, and unified memory spaces, UALink prioritizes direct, high-throughput data transfers between accelerators. UALink is effective for workloads requiring large data transfer and synchronization. Each UALink port typically provides bandwidth up to 100 GB/s via a standard 4-lane configuration.

UALink 1.0 [76, 388], introduced in early 2025, shares architectural similarities with GPU-centric interconnects such as NVLink. However, it is designed as an open standard to support diverse accelerators beyond vendor-specific GPUs (e.g., NVIDIA GPUs). UALink utilizes a single-hop Clos switched topology, establishing dedicated, low-latency communication paths among accelerators, theoretically supporting clusters of up to 1,024 accelerators. As illustrated in Figure 39, this topology simplifies interconnect structure, reducing intra-rack latency to sub-microsecond levels ($<1$ $\mu$s [389]). By minimizing complexity and enhancing scalability, UALink can address stringent synchronization and high-throughput communication requirements common in densely

Table 3: Technical specification comparison of CXL, UALink, and NVLink (Extended).

| Specification | CXL 3.0 | UALink 1.0 | NVLink 5.0 |
|---|---|---|---|
| Unidirectional Bandwidth (GB/s) | 128 (x16 lanes per link, PCIe 6.0) | 100 (x4 lanes per link) | 50 (x2 lanes per link) |
| Latency | Hundreds of ns (100–250 ns typical) | Sub-$\mu$s (<1 $\mu$s within rack) | Sub-500 ns (<500 ns within rack) |
| Flit Size | 256B (PBR), 68B (HBR) | 640B | 48B∼272B |
| Cache Coherency | Yes (hardware-level) | No | No (only support by NVLink C2C) |
| Memory Pooling | Yes | No (only within UALink connected accelerators) | No (only within NVLink connected GPUs) |
| Topology | Point-to-point, switched fabric (various topologies) | Point-to-point, switched fabric (only single-hop Clos) | Point-to-point, switched fabric (may only single-hop Clos) |
| Scalability | Up to 4096 devices | Up to 1024 accelerators | Up to 576 GPUs |
| Typical Deployment Scale | Rack or multi-rack scale | Intra-rack clusters | GPU-node or GPU-cluster scale |
| Use-case / Primary Workload | Memory disaggregation, coherent memory pooling | Accelerator-to-accelerator collective transfers | GPU tensor exchanges, gradient synchronization |
| Consortium | CXL Consortium | UALink Consortium | NVIDIA |
| Interoperability | Open industry standard | Ethernet-based openness | Proprietary (partial openness via NVLink Fusion) |
| Initial Release (Year) | CXL 1.0 (2019) | UALink 0.49 (2024) | NVLink 1.0 (2016) |
| Current Version (Year) | CXL 3.0 (2022), CXL 3.2 (2024) | UALink 1.0 (2025) | NVLink 5.0 (2024), Fusion (2025) |

interconnected accelerator environments.

To maximize throughput, UALink employs 640B data link flits optimized for large data transfers. Although it also supports instruction-level memory access mechanisms, these primarily serve auxiliary management and control functions rather than core high-throughput data operations. Furthermore, UALink explicitly operates as a *non-coherent protocol*; it does not inherently support cache coherence or coherent memory transactions. This design clearly differentiates UALink from CXL, which primarily targets frequent data transactions, coherent memory sharing, cache coherence management, and resource disaggregation.

UALink leverages Ethernet-based topologies tailored for large-scale data transfers and collective communication patterns prevalent in distributed computing environments, such as all-gather operations. Historically, these communication patterns have been extensively used in distributed systems, preceding their adoption in contemporary multi-GPU and accelerator-rich infrastructures [390, 391]. To achieve maximum bandwidth and precise data alignment, UALink Ethernet interfaces adopt optimized frame structures, reduced protocol overhead, and hardware-level synchronization mechanisms engineered for accelerator synchronization requirements [76, 78].

**NVLink and NVLink Fusion.** NVLink is also an interconnect optimized for GPU-to-GPU communication, offering high bandwidth and low latency within GPU-centric data center environments. Introduced before UALink, NVLink enhanced GPU-to-GPU data transfer performance over existing standards (e.g., PCIe). Since its initial release in 2014, NVLink has undergone multiple generational updates, with NVLink 5.0 being the latest version introduced in 2024. NVLink's characteristics are beneficial for deep learning training workloads and HPC applications when deployed with NVIDIA GPUs.

Specifically, NVLink 5.0 provides 50 GB/s of unidirectional bandwidth per link, resulting in 100 GB/s of total bidirectional bandwidth per link [81]. Deployed primarily through NVIDIA's proprietary NVSwitch crossbar, NVLink efficiently supports configurations ranging from tens of GPUs (e.g., NVLink72 [39, 47, 48]) to larger setups utilizing inter-rack network components (e.g., NVLink576 with long-distance network elements). Despite supporting larger-scale deployments, NVLink targets GPU-node or GPU-cluster scales rather than broader rack-level or multi-rack scenarios. Similar to UALink, NVLink employs single-hop Clos (full-mesh) topologies, minimizing latency for critical collective operations such as All-Reduce and All-Gather communications used in transformer-based model training. NVLink 5.0 achieves low-latency communication of less than 500 ns [392].

In contrast to UALink, NVLink utilizes a smaller 48B∼272B flit[4] [393] optimized for efficient transfer of moderate-sized tensors and gradient data. NVLink supports per-node memory-region unification rather than protocol-level cache coherence, simplifying programming complexity and reducing software overhead at the node-level. Historically, NVLink interoperability has been restricted primarily to NVIDIA products, complicating integration into heterogeneous systems.

To address this limitation, NVLink Fusion [82, 83] has been recently introduced, improving interoperability by enabling connections to external processors such as CPUs, NPUs [394–397], and AI-specific processors [398–

[4]While the actual flit size in NVLink is 16B (128-bit), each packet is composed of one header flit followed by up to 16 data flits. The minimum transmission unit consists of 2 data flits (32B), and the maximum includes 16 data flits (256B), yielding total packet sizes between 48B and 272B. In this section, the term NVLink flit denotes a complete packet constructed in this format.

402]. NVLink Fusion provides two primary components: i) a short-reach C2C interface available as coherent IP for external processors, and ii) a Chiplet-based implementation designed for integration with diverse processing units, further optimizing CPU-to-GPU communication.

NVLink Fusion preserves NVLink's intrinsic advantages of high bandwidth and low latency while improving flexibility and shared-memory efficiency between CPUs and GPUs. However, NVLink Fusion is known to require the inclusion of at least one NVIDIA component within interconnected systems [82, 83, 403]. Thus, it currently does not support resource disaggregation or vendor-neutral composability in broader AI infrastructure deployments.

**Comparative summary of CXL and XLink technologies.** Table 3 summarizes the characteristics of the three interconnect technologies, CXL, UALink, and NVLink, discussed in this report.

As described previously, CXL separates compute and memory resources physically, supporting coherent memory pools and cache coherence. As illustrated in the table, CXL typically exhibits lower latency than other interconnect technologies, though specific latencies can vary based on actual implementations. Its cache coherence capability enables accelerators to directly service data from local caches, significantly reducing external data transfers and optimizing performance for data with high locality. CXL connections support up to 256 accelerators (Type 1 or Type 2 devices), while memory-type device connections can scale up to 4,096 endpoints within a single interconnect network. In addition, CXL provides practical PBR routing and switch cascading, greatly enhancing scalability, cache coherence management, and flexible memory allocation. Consequently, CXL is well-suited for composable environments requiring frequent synchronization, intensive memory utilization, and flexible resource configuration.

In contrast, XLink technologies emphasize fast and direct accelerator-to-accelerator data transfers, with a primary focus on high-bandwidth connectivity. While per-transfer latency is generally higher than CXL, XLink technologies can achieve greater aggregate bandwidth by employing larger flit sizes or device-specific optimizations for GPUs or accelerators. Specifically, UALink utilizes Ethernet-based networks, transferring large-scale data optimized for frequent inter-accelerator communication and collective communication patterns. NVLink, however, is tailored to GPU-centric workloads, efficiently exchanging small to medium-sized tensor data or gradients through optimized bandwidth and latency via smaller flit sizes. Both UALink and NVLink assume collective communications through data copying and distributed processing rather than data sharing; thus, neither supports hardware-level cache coherence.

Therefore, combining efficient accelerator communication capabilities of XLink technologies with CXL's flexible memory pooling and cache coherence into a hybrid data center architecture provides a complementary and robust solution. This integration enables accelerators interconnected via UALink or NVLink to leverage CXL-managed memory resources and coherence protocols. Such hybrid architectures can expand the scale-up domain beyond traditional data-parallel approaches, enhancing resource utilization and overall system efficiency.

## 6.2. Integrated Accelerator-Centric, CXL-over-XLink Supercluster Architecture

To accommodate diverse demands of large-scale AI workloads, scaling beyond a single accelerator cluster necessitates efficient inter-cluster communication. Here, the term "cluster" refers to a rack-scale, multi-accelerator system as introduced in the previous data center architecture. Unlike point-to-point intra-cluster networks, inter-cluster connections require more scalable and flexible topologies supporting extensive resource sharing and composability across broader data center infrastructures. CXL can fulfill this requirement through hierarchical, multi-level switching structures, enabling coherent memory pooling among distributed accelerator clusters.

In this subsection, we define a CXL-over-XLink-based *supercluster*, a scalable and hierarchical architecture optimized for accelerator-intensive tasks. A supercluster consists of multiple accelerator clusters interconnected through CXL fabrics. Within each individual accelerator cluster, NVLink or UALink serves as the primary intra-cluster interconnect, providing direct, high-bandwidth, and low-latency communication among accelerators. The detailed configurations of these clusters are described below.

**Accelerator-centric intra-cluster design with UALink and NVLink** Within the CXL-over-XLink-based supercluster architecture, NVLink and UALink serve as intra-cluster interconnect technologies, enabling efficient construction of accelerator clusters. As previously discussed, these interconnect technologies share fundamental design principles, employing single-hop Clos switching topologies optimized for relatively small-scale accelerator clusters, focusing primarily on intra-accelerator communications (cf. Figure 40). Specifically, NVLink supports accelerator clusters composed of multiple GPUs or combined GPU-CPU-memory nodes. NVLink integrates up to 72 GPUs interconnected through multiple NVSwitches, while CPUs within each node are intercon-

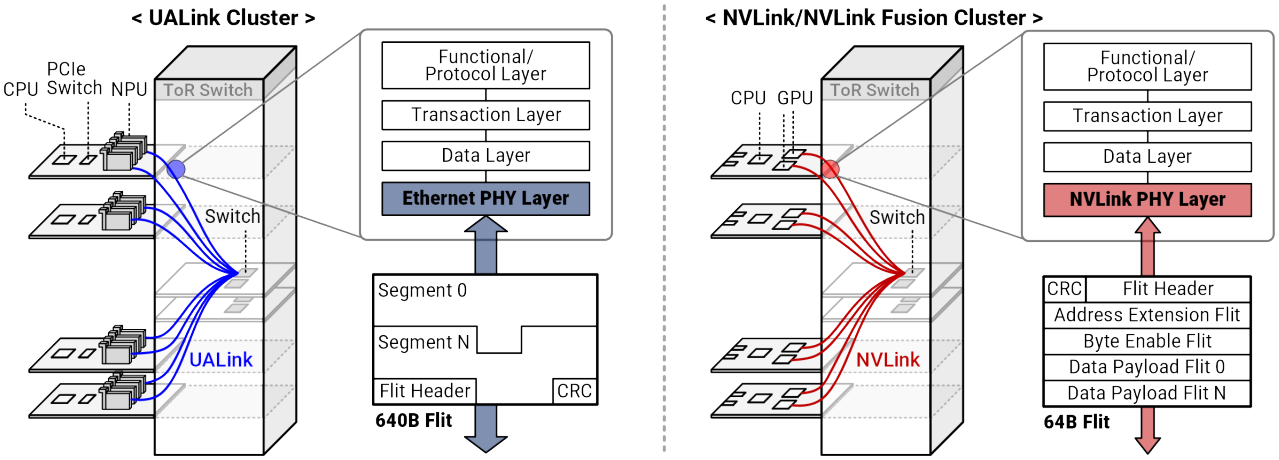



Figure 40: Accelerator-centric intra-cluster design.

nected to GPUs via NVLink C2C interfaces. Similarly, UALink explicitly targets accelerator connectivity within clusters or racks, directly attaching CPU modules to accelerators according to application-specific needs. Accelerators in UALink-based clusters communicate exclusively through UALink switches, theoretically supporting single-hop Clos topologies of up to 1,024 accelerators. This scalability benefits smaller logic-sized, AI-specific accelerators such as NPUs. However, for larger logic-sized accelerators like GPUs, which restrict the number of accelerators per node (e.g., two GPUs per node in GB200/300), the practical deployment scale within a rack closely resembles NVLink configurations (i.e., around 72 accelerators). With UALink, CPUs connect to accelerators through PCIe switches; however, alternative short-reach interconnect solutions such as UCIe [404] may also be employed similarly to NVLink C2C.

Accelerator clusters configured in this manner are likely to consist exclusively of one of the two XLink technologies per cluster, rather than mixing hardware components supported by different interconnects within a single cluster. This limitation arises from fundamental technological differences and interoperability constraints between NVLink and UALink. Specifically, each interconnect employs distinct physical layers and data formats. NVLink uses NVIDIA's proprietary high-speed PHY interfaces with relatively small 48B∼272B flits, whereas UALink adopts Ethernet-based PHY interfaces with significantly larger 640B flits. Thus, we believe that these differences in flit formats, protocol operations, and underlying PHY layers severely limit the feasibility of integrating NVLink and UALink hardware within the same cluster. Therefore, from a strategic interoperability perspective, NVLink requires at least one NVIDIA component (e.g., NVIDIA GPUs), restricting integration with fully third-party accelerator configurations.

Therefore, accelerator clusters employing NVLink and NVSwitches within a CXL-over-XLink supercluster predominantly consist of NVIDIA GPUs, complemented by specialized accelerators optimized for computational tasks not efficiently handled by GPUs. For example, data centers deploying NVIDIA GPUs may integrate accelerators tailored for branch-intensive computations (e.g., tree-based models and conditional logic), workloads with irregular control flows, sparse and irregular memory access patterns (such as graph processing or sparse matrix operations), or latency-critical real-time tasks. Such tasks align poorly with GPU architectures optimized for highly parallel computations. Integrating these heterogeneous accelerators within NVLink-based clusters thus enables data centers to accommodate diverse application demands while maintaining optimized intra-cluster communication performance.

In contrast, UALink-based clusters mainly comprise non-NVIDIA accelerators, such as AMD GPUs or AI-specific processors including Meta's MTIA [398], Amazon's Trainium [399] and Inferentia [400], Microsoft's Maia [401], and Intel's Gaudi [402]. UALink's open, vendor-neutral architecture facilitates diverse accelerator configurations, supporting high-performance intra-cluster communication without dependence on proprietary interfaces. Strategically aligning deployment choices with the interoperability characteristics and architectural strengths of each interconnect ensures optimized intra-cluster performance, enhanced computational throughput, and improved resource efficiency across heterogeneous accelerator environments.

**Scalable inter-cluster communication leveraging CXL.** In a CXL-over-XLink-based supercluster, multiple accelerator clusters interconnected by XLink are integrated into a unified hierarchical architecture through a scalable CXL fabric, forming a large-scale multi-accelerator system. This hybrid interconnect fabric strategy,

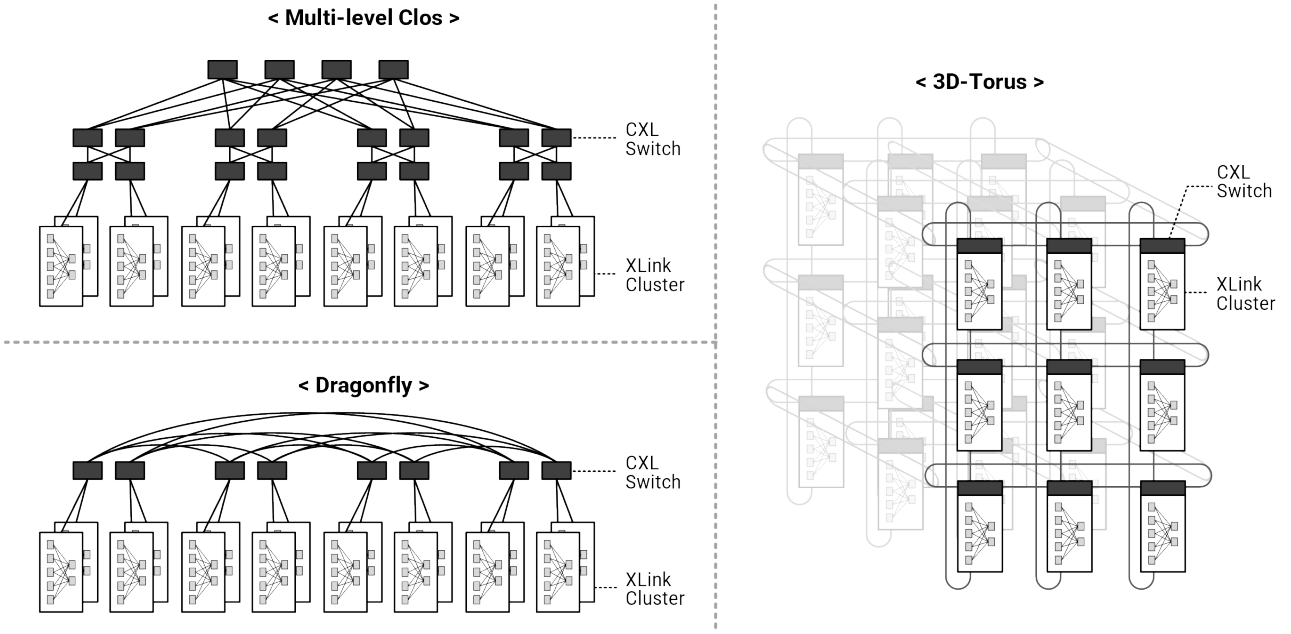


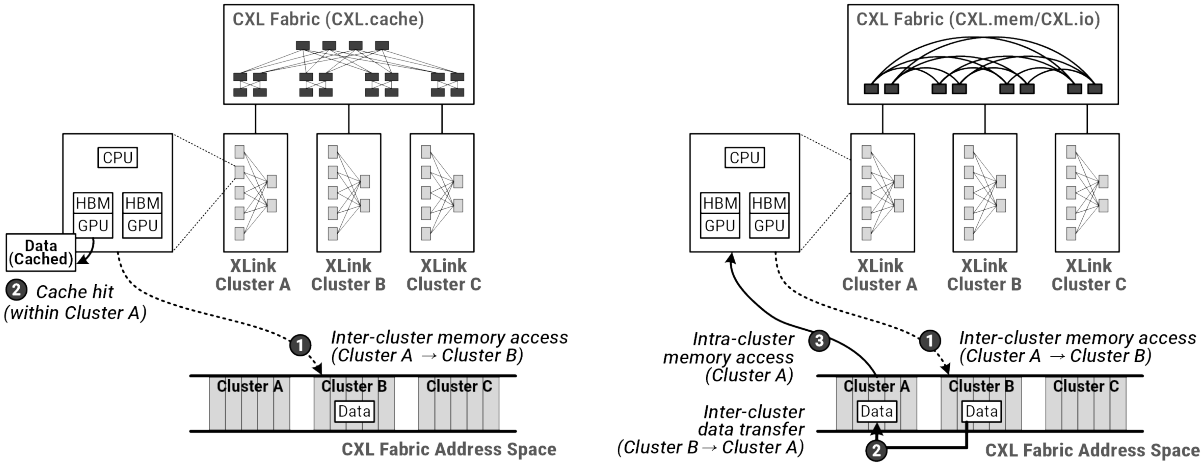

Figure 41: Exemplary CXL-over-XLink supercluster configurations.

as presented in this technical report, can reduce latency and data-transfer overhead, accommodating diverse workload characteristics prevalent in contemporary AI data centers. Specifically, while XLink facilitates rapid intra-cluster data exchanges among accelerators, CXL enables scalable and coherent inter-cluster memory sharing using flexible, hierarchical, switch-based fabric architectures. Unlike the single-hop Clos topologies employed by XLink, CXL supports greater scalability, enabling multiple accelerator clusters to dynamically aggregate distributed memory resources into unified composable pools. This pooled accelerator-local memory reduces the dependency on external storage resources, such as off-chip memory or SSDs, maximizing performance gains and enhancing flexibility and utilization of limited accelerator resources. In addition, CXL resolves interoperability limitations between UALink and NVLink clusters by abstracting each cluster as an independent entity, mediating inter-cluster interactions, thus facilitating seamless coexistence and efficient interaction among heterogeneous accelerator clusters within a unified large-scale architecture.

Figure 41 illustrates exemplary fabric architectures formed by hierarchical CXL switches interconnecting multiple UALink-based and NVLink-based clusters into a unified supercluster. Since CXL supports PBR routing and switch cascading it can be used to implement various topologies such as multi-level Clos, 3D-Torus, and DragonFly, satisfying diverse data center requirements. Moreover, CXL enables devices or clusters to be freely added or removed via hot-plugging, even during large-scale AI data center operations. Consequently, CXL-over-XLink-based supercluster configurations can flexibly adapt to specific workload characteristics, integrating diverse accelerators, memory devices, and computing resources into unified scale-up domains.

Another major advantage of the CXL-over-XLink architecture is its support for inter-cluster protocol-level cache coherence. Leveraging the cache-related sub-protocol (CXL.cache), accelerators within each cluster can directly and coherently access memory resources of other accelerators as well as remote memory resources located in external clusters at instruction-level granularity without software intervention. This approach aggregates local memories of multiple accelerators into a unified memory address space, enabling data to be directly fetched from on-chip accelerator caches. As a result, this minimizes latency and overhead typically associated with conventional inter-node memory access, while also maximizing performance by directly serving localized or shared data from accelerators' own caches. Furthermore, CXL provides dedicated sub-protocol interfaces for memory access (CXL.mem) and high-speed data transactions (CXL.io), being able to support both small-sized instruction-level data transfers and bulk data transfers. These protocols can also be configured to enable direct device-to-device data management without CPU intervention. As illustrated in Figure 42, this hybrid interconnect architecture significantly reduces redundant data movements among accelerators and facilitates computational acceleration across distributed resources, thus maintaining high performance even in memory-intensive workloads.

Finally, CXL-over-XLink can further optimize data movements within superclusters more aggressively. Specifically, CXL.cache-based coherence enables novel paradigms for collective operations (e.g., broadcast, scat-




(a) Protocol-level cache coherence.

(b) Instruction-level data transfer.

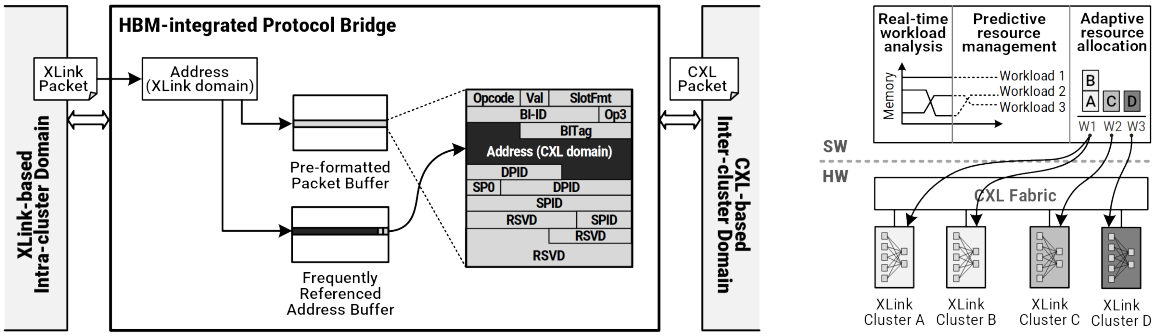
Figure 42: Cache coherence and data movement management in CXL-over-XLink.

ter/gather, and all-reduce), which traditionally incur substantial overhead due to explicit synchronization and redundant data copying. By providing coherent memory access among distributed accelerators, data movements are implicitly managed at the hardware level, enabling accelerators to treat distributed resources as unified memory pools. This fundamentally eliminates overhead associated with explicit synchronization and redundant data copying. This approach not only enhances performance but also simplifies AI model development and management. For example, when programming accelerator kernels, developers can focus exclusively on computational tasks without explicitly managing synchronization or data movement codes. Such software kernels concentrate on parallel computations, while CXL's protocol-level cache coherence policies transparently handle inter-cluster high-speed memory data transfers without any software intervention. Data accesses, particularly those exhibiting locality across diverse workloads, efficiently utilize accelerator-internal caches, maximizing performance and computational efficiency.

**Optimizing hardware and software for integrated XLink and CXL architectures.** Delineating the roles of XLink and CXL within a supercluster architecture provides structural benefits. However, integrating these distinct interconnect technologies presents practical implementation challenges. For example, protocol conversion and data transitions between XLink-based intra-cluster domains and CXL-based inter-cluster domains introduce additional latency due to physical and logical transformations required by differing communication protocols. Such overhead can degrade performance, impacting latency-sensitive workloads. In addition, higher-density accelerator deployments may increase cooling demands, while coherent memory transactions raise concerns about interconnect reliability and error management.

To address these integration challenges, targeted hardware optimizations are required, including specialized system-on-chip bridging interfaces explicitly designed for rapid data-format conversions and streamlined interconnect protocols that minimize latency and reduce handshaking overhead, as shown in Figure 43a. Incorporating HBM within bridging interfaces further mitigates performance penalties arising from inter-domain protocol conversions. In this example, frequently accessed memory addresses or requests can be cached in HBM, preserving pre-converted formats for immediate reuse and thereby eliminating latency overhead during protocol transitions. Intelligent data-placement strategies can be strategically adopted to minimize unnecessary data movements, reducing associated performance impacts.

Aside from these hardware-level optimization approaches, advanced orchestration and software strategies are equally important for improving system performance. Figure 43b illustrates orchestration software frameworks supporting real-time workload monitoring, predictive resource management, and adaptive resource allocation. These capabilities improve operational efficiency and performance in large-scale integrated supercluster architectures. For example, when a specific cluster frequently accesses external memory, orchestration software can physically move the required data closer to the requesting cluster. This relocation allows accelerators within the cluster to efficiently access and process data locally. In addition, fault-tolerance mechanisms such as data redundancy, replication, and memory checkpointing can be implemented through software. These mechanisms




(a) Specialized bridging interfaces with integrated HBM.

(b) Orchestration software.

Figure 43: Optimizing hardware and software for integrated XLink and CXL architectures.

enhance system stability and reliability in large-scale, multi-accelerator environments. Such strategies reduce risks related to component failures, unexpected data corruption, and interconnect disruptions, maintaining stable operation and consistent system performance.

## 6.3. Memory Tiers Leveraging XLink and Lightweight CXL Links

Building upon CXL-over-XLink, we further propose an extended, scalable architecture that integrates a tiered memory hierarchy within supercluster configurations, explicitly designed to address the diverse memory-performance demands of contemporary AI workloads. This structure comprises two distinct memory tiers: i) high-performance local memory managed via XLink and *coherence-centric CXL*, and ii) scalable, composable memory pools enabled through *capacity-oriented CXL*. To effectively establish these memory tiers, we recommend lightweight implementations of CXL specifically tailored for each of these tiers. Finally, we conclude this subsection by introducing data placement and management strategies crucial for strategically leveraging superclusters equipped with these hierarchical memory tiers.

**High-performance accelerator-local memory: XLink with coherence support.** Accelerator clusters within the existing CXL-over-XLink supercluster architecture are interconnected via XLink and utilize high-performance memory technologies, such as HBM or customized high-bandwidth DDR modules. Given that these accelerator structures are pre-defined in each cluster, various HBM modules differing in version or capacity may coexist within a single supercluster. In addition, heterogeneous high-speed memory types may also be intermixed within clusters. As the infrastructure scales, required memory capacities, memory types, and usage patterns differ among clusters, depending on specific workloads and models executed. Since the supercluster architecture already incorporates a CXL fabric for inter-cluster connectivity, this fabric can logically unify distributed high-performance memory resources across clusters into a coherent accelerator-local memory tier.

Within each cluster, accelerator-local memory employs XLink to construct a unified memory address space. Although there can be multiple methods to create this address space, considering the protocol specifications and operational characteristics of XLink, the most fundamental approach is to recognize memory modules of individual accelerators as statically partitioned blocks, forming a unified, linear address space. For instance, UALink can statically partition individual accelerator memory spaces into a unified NUMA-like memory domain across multiple accelerators. Similarly, NVLink constructs a unified address space among multiple devices using virtualization techniques. However, memory unified in this manner does not permit sharing beyond each statically partitioned memory region, requiring software or firmware intervention to explicitly copy data. In addition, due to the absence of protocol-level cache coherence, direct data sharing is infeasible. Consequently, memory accesses targeting regions not locally owned by an accelerator necessitate data transfers via XLink, introducing latency overhead. Such overhead becomes particularly pronounced when accelerators access memory regions across distinct clusters, significantly degrading performance.

To address these performance degradation and latency issues, coherence-centric CXL can be employed. Under the assumption of using cluster-to-cluster CXL connectivity within the proposed CXL-over-XLink architecture without protocol modifications, clusters can designate specific portions of their memory address space and expose them to the inter-cluster CXL fabric in an overlapping, cache-coherent manner, which enable cache

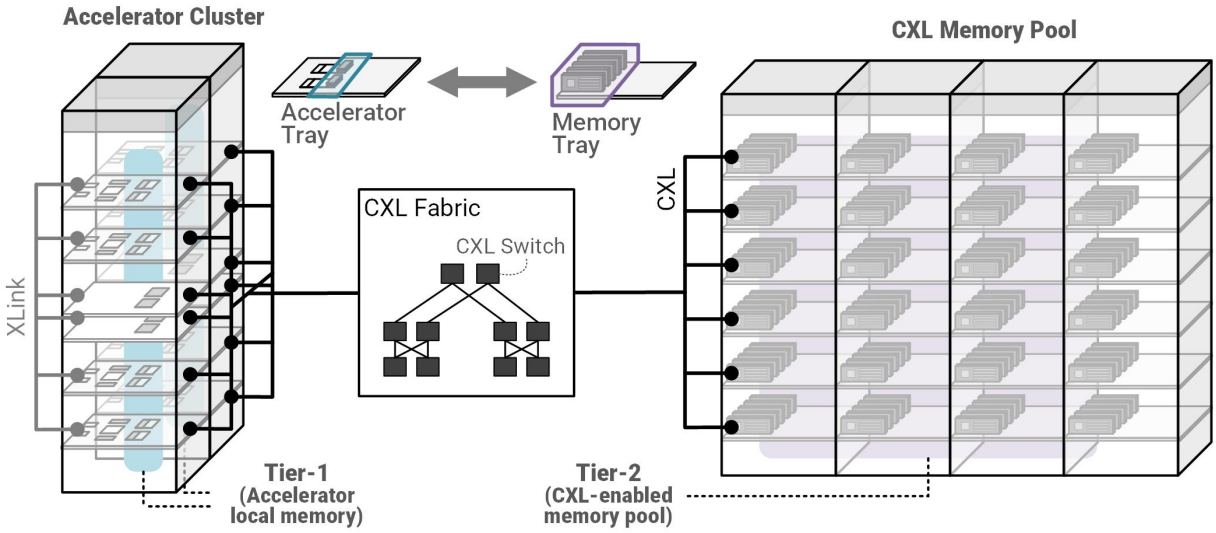



Figure 44: Tiered memory hierarchy configured with CXL memory pool.

coherence selectively for certain applications and datasets. In other words, cache-coherent data sharing becomes possible for designated regions within accelerator-local memory tiers, while other regions continue to be managed through data copying and movement via XLink within a unified address space. Such a partial coherence approach leveraging inter-cluster CXL fabric enhances data locality and performance for specific AI workloads. Moreover, improved data locality confines data accesses within clusters, allowing frequently accessed data to be automatically handled within accelerator on-chip caches, thereby fundamentally eliminating unnecessary data transfers.

For workloads exhibiting high data-sharing and cache coherence requirements, a more advanced and coherence-oriented lightweight application of CXL within clusters can be adopted. In this scenario, dedicated CXL controller logic is integrated into each accelerator, either closely positioned near or directly embedded within the XLink controller. The primary advantage of this configuration is that lightweight, coherence-centric CXL becomes available within clusters alongside XLink, allowing all GPUs or accelerators to perceive a unified memory space and enabling full data sharing. Consequently, explicit collective operations or data movements can be entirely eliminated, resulting in significant performance improvements. Although this design may increase SoC implementation complexity, cost, and the possibility of redundant data transfers, these issues can be effectively mitigated by optimizing the CXL protocol, removing unnecessary protocol features, and focusing explicitly on cache coherence. Such an integrated XLink-CXL controller could be implemented in various ways. However, to eliminate redundancy, bulk data transfers are primarily handled through XLink, while accelerator controllers implement only optimized CXL.cache subprotocols dedicated exclusively to coherence traffic. Despite necessitating detailed controller design and protocol management, this approach ultimately delivers performance benefits, including enhanced cache coherence, simplified data management, and improved computational efficiency across the supercluster.

The combination of accelerator-local memory via XLink and coherence-centric CXL interconnects effectively addresses low-latency and cache-coherence demands at the accelerator-node level. However, modern AI workloads frequently exhibit substantial memory-capacity requirements exceeding the aggregate local memory capacity available at the rack level. Consequently, complementary strategies employing scalable, capacity-oriented composable memory pools, as proposed in the subsequent section, are required to accommodate these extensive memory demands.

**Capacity-oriented composable memory pools: CXL.** Accelerator-local memory serves frequently accessed, performance-critical data. However, modern AI workloads often require significantly larger memory capacities, even at the expense of reduced performance. Representative examples include large-scale embedding tables, caches, and external knowledge bases. To address such challenges, we propose a two-tier composable memory structure integrated within the supercluster architecture, providing flexible memory-capacity expansion.

As illustrated in Figure 44, the proposed two-tier composable memory pools primarily comprise memory

trays physically separated from accelerator clusters and interconnected via a dedicated CXL fabric. In a CXL-over-XLink-based supercluster architecture, tier-1 memory already provides accelerators with a unified memory view managed by coherence-centric CXL and XLink controllers. Therefore, accesses to tier-2 memory pools occur only when memory demands surpass the aggregate accelerator memory capacity available at the rack level. These scenarios are analogous to applications such as RAG, where data retrieval traditionally relies on storage systems or distributed file systems with access latencies ranging from milliseconds to tens of seconds. In contrast, the proposed tier-2 composable memory pools reduce such latency to tens or hundreds of nanoseconds, depending on fabric-switch characteristics. The critical advantage of the proposed approach is capacity scalability, achieved by integrating only memory components within each memory tray, explicitly excluding CPUs or accelerators to maximize memory density and efficiency.

The physical placement of memory trays is related to the number of switch hops and associated latency; thus, these trays can be located anywhere within a CXL-over-XLink-based supercluster, as long as connectivity via CXL fabric is available. Depending on the spatial management requirements of data center designers, memory trays can be flexibly arranged. The tier-2 memory pools' address spaces can be physically configured so that the fabric directly recognizes them, or they can be logically connected through virtual management techniques. Allocating memory trays close to accelerator clusters significantly reduces reliance on slower storage or lower-performance scale-out data access methods. In practice, these external memory resources function as tier-2 capacity-focused memory pools within a hierarchical memory architecture, specifically optimized for memory-capacity expansion.

As in the coherence-centric CXL approach, the proposed capacity-oriented CXL configuration can either utilize the existing CXL-over-XLink fabric or further optimize it to better accommodate large-scale workloads. Given that tier-1 accelerator-local memory already manages cache coherence and handles latency-sensitive data, tier-2 memory pools can be exclusively optimized for memory capacity, simplifying other functionalities and further enhancing cost efficiency. For example, maintaining cache coherence across all memory trays is unnecessary; thus, controllers can be streamlined and efficiency maximized by disabling CXL.cache or CXL.io protocols at switches and endpoints. In particular, when tier-1 memory serves as an exclusive cache, tier-2 composable memory pools can potentially omit CXL.mem altogether, using only the CXL.io protocol for bulk data transfers, similar to traditional storage systems. Regardless of the selected approach, data transfers between accelerator-local memory and composable memory pools persist, making it essential to provide sufficient CXL fabric ports to optimize data transfer performance.

Note that additional optimization strategies beyond protocol simplification are possible for tier-2 capacity-oriented memory pools. As discussed in Section 5.1, memory trays can utilize lower-speed DRAM interfaces instead of high-speed DRAM modules to reduce cost. Alternatively, hybrid memory trays combining high-capacity storage (e.g., flash memory) with modest amounts of HBM can significantly enhance memory capacity while still offering sufficient performance for effective data staging into tier-1 accelerator-local memory. Furthermore, to support the buffering and caching functions of tier-1 accelerator-local memory across extended physical distances, spanning multiple floors or even buildings, the supercluster's memory tiering structure can leverage optical technologies, such as silicon photonics, in place of PCIe PHY for interconnection via CXL.

**Discussion on hierarchical data placement and management strategies.** As discussed previously, integrating accelerator-local memory (XLink-based) with composable memory pools (CXL-based) provides new opportunities for substantial performance and capacity expansion. However, fully leveraging such hierarchical memory structures may require software frameworks capable of intelligent resource management and efficient data placement. Specifically, integrated management methods supporting dynamic allocation of computational tasks across accelerator clusters and optimized memory allocation within composable CXL pools can enhance resource utilization, balance performance, and improve operational efficiency in modern AI data centers.

To fully exploit the hierarchical memory architecture, carefully designed data placement strategies that align with the performance characteristics of each memory tier are critical. Rather than relying on hardware-based management, implementing these placement strategies through software is advantageous. Specifically, software frameworks can precisely evaluate attributes such as data access frequency and latency sensitivity, enabling refined and adaptable placement decisions. For example, latency-critical and frequently accessed data structures, including activation states, embedding vectors, and attention caches, should be placed within accelerator-local memory tiers connected via XLink. In contrast, datasets that are larger or less sensitive to latency are better suited to reside within capacity-oriented composable memory pools facilitated by CXL. Implementing such tier-specific placement requires software-level monitoring of various runtime information, optimizing data placement according to characteristics, thus maximizing resource utilization and overall performance.

Further enhancing the efficiency of sophisticated data-placement strategies may require the addition of

advanced software-based orchestration frameworks. For instance, predictive data-migration algorithms that dynamically adjust data placement based on anticipated workload patterns and access-frequency variations can be introduced. Advanced caching policies, such as temperature-aware caching that prioritizes data according to access frequency, as well as intelligent machine-learning-based prefetching, further minimize latency and overhead associated with inter-tier data transfers. However, excessively frequent inter-tier data migrations can introduce performance degradation; thus, comprehensive approaches involving targeted hardware optimizations, efficient protocol-conversion interfaces between XLink and CXL, and carefully designed data-migration policies remain essential.

In summary, large-scale AI applications deployed within multi-accelerator systems integrating CXL and XLink can significantly benefit from hierarchical data management approaches. Specifically, real-time inference workloads utilize accelerator-local memory for rapid data processing and latency-critical operations, while large-scale embedding lookups and external data retrieval efficiently leverage capacity-oriented composable memory pools provided by CXL. Therefore, clearly defining the roles of each memory tier and strategically implementing data-placement policies ensures optimized computational performance and resource utilization, effectively satisfying the diverse operational demands of modern AI data centers.

# 7. Conclusion

In this technical report, we systematically explored the limitations of traditional GPU-centric architectures in scaling modern AI workloads, highlighting significant performance bottlenecks related to memory capacity, inter-device communication, and resource allocation. To address these challenges, we introduced a composable and modular data center architecture leveraging Compute Express Link (CXL) technology, which disaggregates and dynamically allocates memory, compute, and accelerator resources according to workload-specific demands.

Our empirical evaluations using diverse AI workloads, including retrieval-augmented generation (RAG), Graph-RAG, deep learning recommendation models (DLRM), and MPI-based scientific simulations, demonstrated substantial performance improvements. Specifically, we observed significant reductions in execution latency, communication overhead, and memory management complexity compared to conventional SSD- and RDMA-based infrastructures. These results illustrate how the coherent memory sharing and dynamic composability features of CXL effectively optimize resource utilization and enhance operational flexibility.

In addition, we investigated hybrid architectures integrating dedicated accelerator-centric interconnect technologies (XLink), such as Ultra Accelerator Link (UALink) and NVIDIA's NVLink, alongside CXL. Our analysis revealed the potential ability showing that combining the complementary strengths of these technologies can further enhance scalability, reduce unnecessary long-distance communication, and optimize performance for latency-sensitive intra-accelerator tasks.

Finally, we discussed critical architectural implications of adopting composable CXL infrastructures in real-world data centers. These include dedicated coherent memory pooling, adaptive data placement, accelerator-centric resource management, and sophisticated centralized monitoring frameworks. Future research should focus on addressing deployment challenges at industrial scales, exploring advanced orchestration techniques, and further refining hybrid interconnect strategies. Such efforts are essential to fully realize the potential of composable CXL-based infrastructures in supporting increasingly complex and demanding AI applications.

# References

[1] J. Hendler, "Avoiding another AI winter," *IEEE Intelligent Systems*, vol. 23, no. 02, pp. 2–4, 2008.

[2] A. Toosi, A. G. Bottino, B. Saboury, E. Siegel, and A. Rahmim, "A Brief History of AI: How to Prevent Another Winter," *PET Clinics*, vol. 16, no. 4, pp. 449–469, 2025.

[3] A. Vaswani, N. Shazeer, N. Parmar, J. Uszkoreit, L. Jones, A. N. Gomez, Ł. Kaiser, and I. Polosukhin, "Attention is all you need," *Advances in neural information processing systems*, vol. 30, 2017.

[4] J. Schmidhuber, "Annotated History of Modern AI and Deep Learning," 2022. [Online]. Available: https://arxiv.org/abs/2212.11279

[5] A. Krizhevsky, I. Sutskever, and G. E. Hinton, "ImageNet Classification with Deep Convolutional Neural Networks," in *Advances in Neural Information Processing Systems*, F. Pereira, C. Burges, L. Bottou, and K. Weinberger, Eds., vol. 25. Curran Associates, Inc., 2012. [Online]. Available: https://proceedings.neurips.cc/paper_files/paper/2012/file/c399862d3b9d6b76c8436e924a68c45b-Paper.pdf

[6] J. Devlin, M.-W. Chang, K. Lee, and K. Toutanova, "BERT: Pre-training of Deep Bidirectional Transformers for Language Understanding," in *Proceedings of the 2019 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies, Volume 1 (Long and Short Papers)*, J. Burstein, C. Doran, and T. Solorio, Eds. Minneapolis, Minnesota: Association for Computational Linguistics, Jun. 2019, pp. 4171–4186. [Online]. Available: https://aclanthology.org/N19-1423/

[7] A. G. et al., "The Llama 3 Herd of Models," 2024. [Online]. Available: https://arxiv.org/abs/2407.21783

[8] I. J. Goodfellow, J. Pouget-Abadie, M. Mirza, B. Xu, D. Warde-Farley, S. Ozair, A. Courville, and Y. Bengio, "Generative Adversarial Nets," in *Advances in Neural Information Processing Systems*, Z. Ghahramani, M. Welling, C. Cortes, N. Lawrence, and K. Weinberger, Eds., vol. 27. Curran Associates, Inc., 2014. [Online]. Available: https://proceedings.neurips.cc/paper_files/paper/2014/file/f033ed80deb0234979a61f95710dbe25-Paper.pdf

[9] M. Andrychowicz, M. Denil, S. Gómez, M. W. Hoffman, D. Pfau, T. Schaul, B. Shillingford, and N. de Freitas, "Learning to learn by gradient descent by gradient descent," in *Advances in Neural Information Processing Systems*, D. Lee, M. Sugiyama, U. Luxburg, I. Guyon, and R. Garnett, Eds., vol. 29. Curran Associates, Inc., 2016. [Online]. Available: https://proceedings.neurips.cc/paper_files/paper/2016/file/fb87582825f9d28a8d42c5e5e5e8b23d-Paper.pdf

[10] K. Chandra, A. Xie, J. Ragan-Kelley, and E. MEIJER, "Gradient Descent: The Ultimate Optimizer," in *Advances in Neural Information Processing Systems*, S. Koyejo, S. Mohamed, A. Agarwal, D. Belgrave, K. Cho, and A. Oh, Eds., vol. 35. Curran Associates, Inc., 2022, pp. 8214–8225. [Online]. Available: https://proceedings.neurips.cc/paper_files/paper/2022/file/36ce475705c1dc6c50a5956cedff3d01-Paper-Conference.pdf

[11] S. Ruder, "An overview of gradient descent optimization algorithms," *arXiv preprint arXiv:1609.04747*, 2016.

[12] L. Bottou, F. E. Curtis, and J. Nocedal, "Optimization methods for large-scale machine learning," *SIAM review*, vol. 60, no. 2, pp. 223–311, 2018.

[13] Y. Bengio, "Practical recommendations for gradient-based training of deep architectures," in *Neural networks: Tricks of the trade: Second edition*. Springer, 2012, pp. 437–478.

[14] G. Montúfar, R. Pascanu, K. Cho, and Y. Bengio, "On the Number of Linear Regions of Deep Neural Networks," in *Advances in Neural Information Processing Systems*, Z. Ghahramani, M. Welling, C. Cortes, N. Lawrence, and K. Weinberger, Eds., vol. 27. Curran Associates, Inc., 2014. [Online]. Available: https://proceedings.neurips.cc/paper_files/paper/2014/file/fa6f2a469cc4d61a92d96e74617c3d2a-Paper.pdf

[15] Y. Bengio, A. Courville, and P. Vincent, "Representation learning: A review and new perspectives," *IEEE transactions on pattern analysis and machine intelligence*, vol. 35, no. 8, pp. 1798–1828, 2013.

[16] A. Ansuini, A. Laio, J. H. Macke, and D. Zoccolan, "Intrinsic dimension of data representations in deep neural networks," *Advances in Neural Information Processing Systems*, vol. 32, 2019.

[17] C. Ruiz, H. Ren, K. Huang, and J. Leskovec, "High dimensional, tabular deep learning with an auxiliary knowledge graph," in *Advances in Neural Information Processing Systems*, A. Oh, T. Naumann, A. Globerson, K. Saenko, M. Hardt, and S. Levine, Eds., vol. 36. Curran Associates, Inc., 2023, pp. 26 348–26 371. [Online]. Available: https://proceedings.neurips.cc/paper_files/paper/2023/file/53dd219b6b11abc8ce523921c18c7a3e-Paper-Conference.pdf

[18] C. Hawthorne, A. Jaegle, C. Cangea, S. Borgeaud, C. Nash, M. Malinowski, S. Dieleman, O. Vinyals, M. Botvinick, I. Simon, H. Sheahan, N. Zeghidour, J.-B. Alayrac, J. Carreira, and J. Engel, "General-purpose, long-context autoregressive modeling with Perceiver AR," in *Proceedings of the 39th International Conference on Machine Learning*, ser. Proceedings of Machine Learning Research, K. Chaudhuri, S. Jegelka, L. Song, C. Szepesvari, G. Niu, and S. Sabato, Eds., vol. 162. PMLR, 17–23 Jul 2022, pp. 8535–8558. [Online]. Available: https://proceedings.mlr.press/v162/hawthorne22a.html

[19] B. Peccerillo, M. Mannino, A. Mondelli, and S. Bartolini, "A survey on hardware accelerators: Taxonomy, trends, challenges, and perspectives," *Journal of Systems Architecture*, vol. 129, p. 102561, 2022. [Online]. Available: https://www.sciencedirect.com/science/article/pii/S1383762122001138

[20] P. M. et al., "The transformational role of GPU computing and deep learning in drug discovery," *Nature Machine Intelligence*, vol. 4, no. 3, pp. 211–221, 2022.

[21] S. Rajbhandari, J. Rasley, O. Ruwase, and Y. He, "ZeRO: Memory optimizations Toward Training Trillion Parameter Models," in *SC20: International Conference for High Performance Computing, Networking, Storage and Analysis*, 2020, pp. 1–16.

[22] J. Rasley, S. Rajbhandari, O. Ruwase, and Y. He, "DeepSpeed: System Optimizations Enable Training Deep Learning Models with Over 100 Billion Parameters," in *Proceedings of the 26th ACM SIGKDD International Conference on Knowledge Discovery & Data Mining*, ser. KDD '20. New York, NY, USA: Association for Computing Machinery, 2020, p. 3505–3506. [Online]. Available: https://doi.org/10.1145/3394486.3406703

[23] S. Sano, Y. Bando, K. Hiwada, H. Kajihara, T. Suzuki, Y. Nakanishi, D. Taki, A. Kaneko, and T. Shiozawa, "Gpu graph processing on cxl-based microsecond-latency external memory," in *Proceedings of the SC'23 Workshops of the International Conference on High Performance Computing, Network, Storage, and Analysis*, 2023, pp. 962–972.

[24] S. W. Min, V. S. Mailthody, Z. Qureshi, J. Xiong, E. Ebrahimi, and W.-m. Hwu, "Emogi: Efficient memory-access for out-of-memory graph-traversal in gpus," *arXiv preprint arXiv:2006.06890*, 2020.

[25] D. Narayanan, M. Shoeybi, J. Casper, P. LeGresley, M. Patwary, V. Korthikanti, D. Vainbrand, P. Kashinkunti, J. Bernauer, B. Catanzaro, A. Phanishayee, and M. Zaharia, "Efficient large-scale language model training on gpu clusters using megatron-lm," in *Proceedings of the International Conference for High Performance Computing, Networking, Storage and Analysis*, ser. SC '21. New York, NY, USA: Association for Computing Machinery, 2021. [Online]. Available: https://doi.org/10.1145/3458817.3476209

[26] W. Kwon, Z. Li, S. Zhuang, Y. Sheng, L. Zheng, C. H. Yu, J. Gonzalez, H. Zhang, and I. Stoica, "Efficient memory management for large language model serving with pagedattention," in *Proceedings of the 29th Symposium on Operating Systems Principles*, ser. SOSP '23. New York, NY, USA: Association for Computing Machinery, 2023, p. 611–626. [Online]. Available: https://doi.org/10.1145/3600006.3613165

[27] Z. Zhang, Y. Sheng, T. Zhou, T. Chen, L. Zheng, R. Cai, Z. Song, Y. Tian, C. Ré, C. Barrett, Z. A. Wang, and B. Chen, "H2O: Heavy-hitter oracle for efficient generative inference of large language models," in *Advances in Neural Information Processing Systems*, A. Oh, T. Naumann, A. Globerson, K. Saenko, M. Hardt, and S. Levine, Eds., vol. 36. Curran Associates, Inc., 2023, pp. 34 661–34 710. [Online]. Available: https://proceedings.neurips.cc/paper_files/paper/2023/file/6ceefa7b15572587b78ecfcebb2827f8-Paper-Conference.pdf

[28] M. Adnan, A. Arunkumar, G. Jain, P. J. Nair, I. Soloveychik, and P. Kamath, "Keyformer: Kv cache reduction through key tokens selection for efficient generative inference," in *Proceedings of Machine Learning and Systems*, P. Gibbons, G. Pekhimenko, and C. D. Sa, Eds., vol. 6, 2024, pp. 114–127. [Online]. Available: https://proceedings.mlsys.org/paper_files/paper/2024/file/48fecef47b19fe501d27d338b6d52582-Paper-Conference.pdf

[29] B. Li, X. Wang, J. Wang, Y. Liu, Y. Gong, H. Lu, W. Dang, W. Zhang, X. Huang, M. Chen, J. Chen, C. He, Y. Liu, X. Hu, C. Liu, X. Ji, Y. Xia, X. Li, Z. He, Y. Wang, and X. Zou, "TCCL: Co-optimizing Collective Communication and Traffic Routing for GPU-centric Clusters," in *Proceedings of the 2024 SIGCOMM Workshop on Networks for AI Computing*, ser. NAIC '24. New York, NY, USA: Association for Computing Machinery, 2024, p. 48–53. [Online]. Available: https://doi.org/10.1145/3672198.3673799

[30] Y. Jin, C.-F. Wu, D. Brooks, and G.-Y. Wei, "S^3: Increasing GPU Utilization during Generative Inference for Higher Throughput," in *Advances in Neural Information Processing Systems*, A. Oh, T. Naumann, A. Globerson, K. Saenko, M. Hardt, and S. Levine, Eds., vol. 36. Curran Associates, Inc., 2023, pp. 18 015–18 027. [Online]. Available: https://proceedings.neurips.cc/paper_files/paper/2023/file/3a13be0c5dae69e0f08065f113fb10b8-Paper-Conference.pdf

[31] NVIDIA, "NVIDIA Grace GPU." [Online]. Available: https://www.nvidia.com/en-us/data-center/grace-cpu-superchip/

[32] Q. Hu, P. Sun, S. Yan, Y. Wen, and T. Zhang, "Characterization and prediction of deep learning workloads in large-scale GPU datacenters," in *Proceedings of the International Conference for High Performance Computing, Networking, Storage and Analysis*, ser. SC '21. New York, NY, USA: Association for Computing Machinery, 2021. [Online]. Available: https://doi.org/10.1145/3458817.3476223

[33] F. Werner, M. Weisgut, and T. Rabl, "Towards Memory Disaggregation via NVLink C2C: Benchmarking CPU-Requested GPU Memory Access," in *Proceedings of the 4th Workshop on Heterogeneous Composable and Disaggregated Systems*, ser. HCDS '25. New York, NY, USA: Association for Computing Machinery, 2025, p. 8–14. [Online]. Available: https://doi.org/10.1145/3723851.3723853

[34] H. Zhang, Z. Zheng, S. Xu, W. Dai, Q. Ho, X. Liang, Z. Hu, J. Wei, P. Xie, and E. P. Xing, "Poseidon: An Efficient Communication Architecture for Distributed Deep Learning on GPU Clusters," in *2017 USENIX Annual Technical Conference (USENIX ATC 17)*. Santa Clara, CA: USENIX Association, Jul. 2017, pp. 181–193. [Online]. Available: https://www.usenix.org/conference/atc17/technical-sessions/presentation/zhang

[35] F. V. Zacarias, K. Palli, S. Vazhkudai, and E. Grevelink, "A memory perspective: The effects of finetuning llms with highbandwidth memory," in *Micron*, 2024.

[36] NVIDIA, "Whitepaper: NVDIA GH200 Grace Hopper Superchip Architecture," 2023.

[37] H. Miao, M. Jeon, G. Pekhimenko, K. S. McKinley, and F. X. Lin, "Streambox-hbm: Stream analytics on high bandwidth hybrid memory," in *Proceedings of the Twenty-Fourth International Conference on Architectural Support for Programming Languages and Operating Systems*, 2019, pp. 167–181.

[38] M. Zhu, Y. Zhuo, C. Wang, W. Chen, and Y. Xie, "Performance evaluation and optimization of HBM-enabled GPU for data-intensive applications," *IEEE Transactions on Very Large Scale Integration (VLSI) Systems*, vol. 26, no. 5, pp. 831–840, 2018.

[39] NVIDIA, "NVIDIA GB200 NVL72 Delivers Trillion-Parameter LLM Training and Real-Time Inference." [Online]. Available: https://developer.nvidia.com/blog/nvidia-gb200-nvl72-delivers-trillion-parameter-llm-training-and-real-time-inference/

[40] W. Fedus, B. Zoph, and N. Shazeer, "Switch transformers: Scaling to trillion parameter models with simple and efficient sparsity," *Journal of Machine Learning Research*, vol. 23, no. 120, pp. 1–39, 2022. [Online]. Available: http://jmlr.org/papers/v23/21-0998.html

[41] K. Guu, K. Lee, Z. Tung, P. Pasupat, and M. Chang, "Retrieval augmented language model pre-training," in *Proceedings of the 37th International Conference on Machine Learning*, ser. Proceedings of Machine Learning Research, H. D. III and A. Singh, Eds., vol. 119. PMLR, 13–18 Jul 2020, pp. 3929–3938. [Online]. Available: https://proceedings.mlr.press/v119/guu20a.html

[42] S. e. a. Borgeaud, "Improving language models by retrieving from trillions of tokens," in *Proceedings of the 39th International Conference on Machine Learning*, ser. Proceedings of Machine Learning Research, K. Chaudhuri, S. Jegelka, L. Song, C. Szepesvari, G. Niu, and S. Sabato, Eds., vol. 162. PMLR, 17–23 Jul 2022, pp. 2206–2240. [Online]. Available: https://proceedings.mlr.press/v162/borgeaud22a.html

[43] M. Shoeybi, M. Patwary, R. Puri, P. LeGresley, J. Casper, and B. Catanzaro, "Megatron-lm: Training multi-billion parameter language models using model parallelism," 2020. [Online]. Available: https://arxiv.org/abs/1909.08053

[44] A. Roberts, C. Raffel, and N. Shazeer, "How much knowledge can you pack into the parameters of a language model?" 2020. [Online]. Available: https://arxiv.org/abs/2002.08910

[45] D. Narayanan, A. Harlap, A. Phanishayee, V. Seshadri, N. R. Devanur, G. R. Ganger, P. B. Gibbons, and M. Zaharia, "Pipedream: generalized pipeline parallelism for dnn training," in *Proceedings of the 27th ACM Symposium on Operating Systems Principles*, ser. SOSP '19. New York, NY, USA: Association for Computing Machinery, 2019, p. 1–15. [Online]. Available: https://doi.org/10.1145/3341301.3359646

[46] J. Howarth, "Number of parameters in gpt-4 (latest data)," 2025. [Online]. Available: https://explodingtopics.com/blog/gpt-parameters

[47] NVIDIA, "NVIDIA GB200 NVL72." [Online]. Available: https://www.nvidia.com/en-us/data-center/gb200-nvl72

[48] NVIDIA, "NVIDIA GB300 NVL72." [Online]. Available: https://www.nvidia.com/en-us/data-center/gb300-nvl72/

[49] NVIDIA, "NVIDIA DGX GB300." [Online]. Available: https://www.nvidia.com/en-us/data-center/dgx-gb300/

[50] W. Li, X. Liu, Y. Li, Y. Jin, H. Tian, Z. Zhong, G. Liu, Y. Zhang, and K. Chen, "Understanding communication characteristics of distributed training," in *Proceedings of the 8th Asia-Pacific Workshop on Networking*, ser. APNet '24. New York, NY, USA: Association for Computing Machinery, 2024, p. 1–8. [Online]. Available: https://doi.org/10.1145/3663408.3663409

[51] S. Hsia, A. Golden, B. Acun, N. Ardalani, Z. DeVito, G.-Y. Wei, D. Brooks, and C.-J. Wu, "MAD-Max Beyond Single-Node: Enabling Large Machine Learning Model Acceleration on Distributed Systems," in *2024 ACM/IEEE 51st Annual International Symposium on Computer Architecture (ISCA)*. IEEE, 2024, pp. 818–833.

[52] L.-W. Chang, W. Bao, Q. Hou, C. Jiang, N. Zheng, Y. Zhong, X. Zhang, Z. Song, C. Yao, Z. Jiang *et al.*, "Flux: fast software-based communication overlap on gpus through kernel fusion," *arXiv preprint arXiv:2406.06858*, 2024.

[53] NVIDIA, "Collective communication library NCCL," 2021. [Online]. Available: https://developer.nvidia.com/nccl

[54] C. Jiang, Y. Tian, Z. Jia, S. Zheng, C. Wu, and Y. Wang, "Lancet: Accelerating mixture-of-experts training via whole graph computation-communication overlapping," *Proceedings of Machine Learning and Systems*, vol. 6, pp. 74–86, 2024.

[55] G. Huang, H. Li, L. Qin, J. Huang, Y. Kang, Y. Ding, and Y. Xie, "Traci: Network acceleration of input-dynamic communication for large-scale deep learning recommendation model," in *Proceedings of the 52nd Annual International Symposium on Computer Architecture*, ser. ISCA '25. New York, NY, USA: Association for Computing Machinery, 2025, p. 1880–1893. [Online]. Available: https://doi.org/10.1145/3695053.3731105

[56] S. Sano, Y. Bando, K. Hiwada, H. Kajihara, T. Suzuki, Y. Nakanishi, D. Taki, A. Kaneko, and T. Shiozawa, "GPU graph processing on cxl-based microsecond-latency external memory," in *Proceedings of the SC'23 Workshops of the International Conference on High Performance Computing, Network, Storage, and Analysis*, 2023, pp. 962–972.

[57] W. Hou, J. Zhang, Z. Wang, and M. Liu, "Understanding routable {PCIe} performance for composable infrastructures," in *21st USENIX Symposium on Networked Systems Design and Implementation (NSDI 24)*, 2024, pp. 297–312.

[58] R. Neugebauer, G. Antichi, J. F. Zazo, Y. Audzevich, S. López-Buedo, and A. W. Moore, "Understanding PCIe performance for end host networking," in *Proceedings of the 2018 Conference of the ACM Special Interest Group on Data Communication*, 2018, pp. 327–341.

[59] CXL Consortium, "CXL 1.0 specification," 2019. [Online]. Available: https://computeexpresslink.org/wp-content/uploads/2024/02/CXL-1.0-Specification.pdf

[60] CXL Consortium, "CXL 2.0 specification," 2020. [Online]. Available: https://computeexpresslink.org/wp-content/uploads/2024/02/CXL-2.0-Specification.pdf

[61] CXL Consortium, "CXL 3.2 specification," 2024. [Online]. Available: https://computeexpresslink.org/cxl-specification/

[62] S.-P. Yang, M. Kim, S. Nam, J. Park, J. yong Choi, E. H. Nam, E. Lee, S. Lee, and B. S. Kim, "Overcoming the memory wall with CXL-Enabled SSDs," in *2023 USENIX Annual Technical Conference (USENIX ATC 23)*. Boston, MA: USENIX Association, Jul. 2023, pp. 601–617. [Online]. Available: https://www.usenix.org/conference/atc23/presentation/yang-shao-peng

[63] D. Das Sharma, R. Blankenship, and D. Berger, "An introduction to the compute express link (cxl) interconnect," *ACM Comput. Surv.*, vol. 56, no. 11, Jul. 2024. [Online]. Available: https://doi.org/10.1145/3669900

[64] M. Jung, "Hello bytes, bye blocks: Pcie storage meets compute express link for memory expansion (cxl-ssd)," in *Proceedings of the 14th ACM Workshop on Hot Topics in Storage and File Systems*, 2022, pp. 45–51.

[65] H. Li, D. S. Berger, L. Hsu, D. Ernst, P. Zardoshti, S. Novakovic, M. Shah, S. Rajadnya, S. Lee, I. Agarwal, M. D. Hill, M. Fontoura, and R. Bianchini, "Pond: CXL-Based Memory Pooling Systems for Cloud Platforms," in *Proceedings of the 28th ACM International Conference on Architectural Support for Programming Languages and Operating Systems, Volume 2*, ser. ASPLOS 2023. New York, NY, USA: Association for Computing Machinery, 2023, p. 574–587. [Online]. Available: https://doi.org/10.1145/3575693.3578835

[66] A. Godbole, "Breaking the Memory Wall with Compute Express Link (CXL)," 2024. [Online]. Available: https://community.intel.com/t5/Blogs/Tech-Innovation/Data-Center/Breaking-the-Memory-Wall-with-Compute-Express-Link-CXL/post/1594848

[67] D. Gouk, S. Lee, M. Kwon, and M. Jung, "Direct access,{High-Performance} memory disaggregation with {DirectCXL}," in *2022 USENIX Annual Technical Conference (USENIX ATC 22)*, 2022, pp. 287–294.

[68] P. Lewis, E. Perez, A. Piktus, F. Petroni, V. Karpukhin, N. Goyal, H. Küttler, M. Lewis, W.-t. Yih, T. Rocktäschel, S. Riedel, and D. Kiela, "Retrieval-Augmented Generation for Knowledge-Intensive NLP Tasks," in *Advances in Neural Information Processing Systems*, H. Larochelle, M. Ranzato, R. Hadsell, M. Balcan, and H. Lin, Eds., vol. 33. Curran Associates, Inc., 2020, pp. 9459–9474. [Online]. Available: https://proceedings.neurips.cc/paper_files/paper/2020/file/6b493230205f780e1bc26945df7481e5-Paper.pdf

[69] Z. Jiang, F. Xu, L. Gao, Z. Sun, Q. Liu, J. Dwivedi-Yu, Y. Yang, J. Callan, and G. Neubig, "Active retrieval augmented generation," in *Proceedings of the 2023 Conference on Empirical Methods in Natural Language Processing*, H. Bouamor, J. Pino, and K. Bali, Eds. Singapore: Association for Computational Linguistics, Dec. 2023, pp. 7969–7992. [Online]. Available: https://aclanthology.org/2023.emnlp-main.495/

[70] J. Chen, H. Lin, X. Han, and L. Sun, "Benchmarking large language models in retrieval-augmented generation," in *Proceedings of the AAAI Conference on Artificial Intelligence*, vol. 38, no. 16, 2024, pp. 17 754–17 762.

[71] D. Edge, H. Trinh, N. Cheng, J. Bradley, A. Chao, A. Mody, S. Truitt, D. Metropolitansky, R. O. Ness, and J. Larson, "From local to global: A graph rag approach to query-focused summarization," *arXiv preprint arXiv:2404.16130*, 2024.

[72] T. Zhang, J. Yi, Z. Xu, and A. Shrivastava, "KV Cache is 1 Bit Per Channel: Efficient Large Language Model Inference with Coupled Quantization," in *Advances in Neural Information Processing Systems*, A. Globerson, L. Mackey, D. Belgrave, A. Fan, U. Paquet, J. Tomczak, and C. Zhang, Eds., vol. 37. Curran Associates, Inc., 2024, pp. 3304–3331. [Online]. Available: https://proceedings.neurips.cc/paper_files/paper/2024/file/05d6b5b6901fb57d2c287e1d3ce6d63c-Paper-Conference.pdf

[73] W. Lee, J. Lee, J. Seo, and J. Sim, "InfiniGen: Efficient Generative Inference of Large Language Models with Dynamic KV Cache Management," in *18th USENIX Symposium on Operating Systems Design and Implementation (OSDI 24)*. Santa Clara, CA: USENIX Association, Jul. 2024, pp. 155–172. [Online]. Available: https://www.usenix.org/conference/osdi24/presentation/lee

[74] Y. Liu, H. Li, Y. Cheng, S. Ray, Y. Huang, Q. Zhang, K. Du, J. Yao, S. Lu, G. Ananthanarayanan, M. Maire, H. Hoffmann, A. Holtzman, and J. Jiang, "CacheGen: KV Cache Compression and Streaming for Fast Large Language Model Serving," in *Proceedings of the ACM SIGCOMM 2024 Conference*, ser. ACM SIGCOMM '24. New York, NY, USA: Association for Computing Machinery, 2024, p. 38–56. [Online]. Available: https://doi.org/10.1145/3651890.3672274

[75] Z. Liu, A. Desai, F. Liao, W. Wang, V. Xie, Z. Xu, A. Kyrillidis, and A. Shrivastava, "Scissorhands: Exploiting the Persistence of Importance Hypothesis for LLM KV Cache Compression

at Test Time," in *Advances in Neural Information Processing Systems*, A. Oh, T. Naumann, A. Globerson, K. Saenko, M. Hardt, and S. Levine, Eds., vol. 36. Curran Associates, Inc., 2023, pp. 52 342–52 364. [Online]. Available: https://proceedings.neurips.cc/paper_files/paper/2023/file/a452a7c6c463e4ae8fbdc614c6e983e6-Paper-Conference.pdf

[76] UALink Consortium, "UALink 200G 1.0 Specification," 2025. [Online]. Available: https://ualinkconsortium.org/wp-content/uploads/2025/04/UALink-1.0-White_Paper_FINAL.pdf

[77] UALink Consortium. [Online]. Available: https://ualinkconsortium.org/

[78] J. Ames and R. Lowman, "How Ultra Ethernet and UALink Enable High-Performance, Scalable AI Networks," 2025. [Online]. Available: https://www.synopsys.com/articles/ultra-ethernet-ualink-ai-networks.html#5

[79] NVIDIA, "What is NVLink?" 2023. [Online]. Available: https://blogs.nvidia.com/blog/what-is-nvidia-nvlink

[80] NVIDIA, "NVLink NVIDIA." [Online]. Available: https://www.nvidia.com/en-us/design-visualization/nvlink-bridges

[81] NVIDIA, "WNVIDIA NVLink and NVLink Switch," 2023. [Online]. Available: https://www.nvidia.com/en-us/data-center/nvlink/

[82] NVIDIA, "NVIDIA Fusion," 2025. [Online]. Available: https://www.nvidia.com/en-us/data-center/nvlink-fusion/

[83] A. Sharpiro, "NVIDIA Unveils NVLink Fusion for Industry to Build Semi-Custom AI Infrastructure With NVIDIA Partner Ecosystem," 2025. [Online]. Available: https://nvidianews.nvidia.com/news/nvidia-nvlink-fusion-semi-custom-ai-infrastructure-partner-ecosystem

[84] A. K. anda Michael Kaminsky and D. G. Anderson, "Design guidelines for high performance RDMA systems," in *2016 USENIX annual technical conference (USENIX ATC 16)*. Denver, CO: USENIX Association, 2016. [Online]. Available: https://www.usenix.org/system/files/conference/atc16/atc16_paper-kalia.pdf

[85] J. Wang, B. Lin, J. Zhang, M. Sun, and Y. Pan, "An optimized rdma qp communication mechanism for hyperscale ai infrastructure," *Cluster Computing*, vol. 28, 2024.

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

[87] A. Gangidi, R. Miao, S. Zheng, S. J. Bondu, G. Goes, H. Morsy, R. Puri, M. Riftadi, A. J. Shetty, J. Yang, S. Zhang, M. J. Fernandez, S. Gandham, and H. Zeng, "Rdma over ethernet for distributed training at meta scale," in *Proceedings of the ACM SIGCOMM 2024 Conference*, ser. ACM SIGCOMM '24. New York, NY, USA: Association for Computing Machinery, 2024, p. 57–70. [Online]. Available: https://doi.org/10.1145/3651890.3672233

[88] Graph500, "Graph 500 benchmarks," https://graph500.org/.

[89] NASA Advanced Supercomputing Division, "NASA Parallel Benchmarks," https://www.nas.nasa.gov/software/npb.html.

[90] Lattice Boltzmann Method, "Lattice Boltzmann Method Benchmarks," https://www.spec.org/cpu2017/Docs/benchmarks/619.lbm_s.html.

[91] S. McIntosh-Smith, M. Martineau, T. Deakin, G. Pawelczak, W. Gaudin, P. Garrett, W. Liu, R. Smedley-Stevenson, and D. Beckingsale, "Tealeaf: A mini-application to enable design-space explorations for iterative sparse linear solvers," in *2017 IEEE International Conference on Cluster Computing (CLUSTER)*. IEEE, 2017, pp. 842–849.

[92] Łach and D. Svyetlichnyy, "Advances in Numerical Modeling for Heat Transfer and Thermal Management: A Review of Computational Approaches and Environmental Impacts," *Energies*, vol. 18, no. 5, 2025. [Online]. Available: https://www.mdpi.com/1996-1073/18/5/1302

[93] I. Sutskever, O. Vinyals, and Q. V. Le, "Sequence to sequence learning with neural networks," in *Proceedings of the 28th International Conference on Neural Information Processing Systems - Volume 2*, ser. NIPS'14. Cambridge, MA, USA: MIT Press, 2014, p. 3104–3112.

[94] C.-C. Chiu, T. N. Sainath, Y. Wu, R. Prabhavalkar, P. Nguyen, Z. Chen, A. Kannan, R. J. Weiss, K. Rao, E. Gonina, N. Jaitly, B. Li, J. Chorowski, and M. Bacchiani, "State-of-the-Art Speech Recognition with Sequence-to-Sequence Models," in *2018 IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP)*, 2018, pp. 4774–4778.

[95] M.-T. Luong, Q. V. Le, I. Sutskever, O. Vinyals, and L. Kaiser, "Multi-task Sequence to Sequence Learning," 2016. [Online]. Available: https://arxiv.org/abs/1511.06114

[96] A. Radford, K. Narasimhan, T. Salimans, I. Sutskever *et al.*, "Improving language understanding by generative pre-training," 2018.

[97] OpenAI, "New models and developer products announced at devday," 2023. [Online]. Available: https://openai.com/index/new-models-and-developer-products-announced-at-devday/

[98] Meta, "Models. Llama3.1, 3.2, 3.3, Llama4." [Online]. Available: https://www.llama.com/

[99] OpenAI, "GPT-4 Turbo," 2025. [Online]. Available: https://platform.openai.com/docs/models/gpt-4-turbo

[100] K. Kavukcuoglu, "Gemini 2.5: Our most intelligent AI model," 2025. [Online]. Available: https://blog.google/technology/google-deepmind/gemini-model-thinking-updates-march-2025/#gemini-2-5-thinking

[101] A. Thompson, "Report: Google DeepMind Gemini," 2024. [Online]. Available: https://lifearchitect.ai/gemini-report/

[102] A. Graves, "Generating sequences with recurrent neural networks," *arXiv preprint arXiv:1308.0850*, 2013.

[103] K. Cho, B. van Merrienboer, C. Gulcehre, D. Bahdanau, F. Bougares, H. Schwenk, and Y. Bengio, "Learning phrase representations using rnn encoder-decoder for statistical machine translation," 2014. [Online]. Available: https://arxiv.org/abs/1406.1078

[104] A. Graves, "Generating sequences with recurrent neural networks," *arXiv preprint arXiv:1308.0850*, 2013.

[105] A. Graves, "Long short-term memory," *Supervised sequence labelling with recurrent neural networks*, pp. 37–45, 2012.

[106] B. e. a. Tom, "Language Models are Few-Shot Learners," in *Advances in Neural Information Processing Systems*, H. Larochelle, M. Ranzato, R. Hadsell, M. Balcan, and H. Lin, Eds., vol. 33. Curran Associates, Inc., 2020, pp. 1877–1901. [Online]. Available: https://proceedings.neurips.cc/paper_files/paper/2020/file/1457c0d6bfcb4967418bfb8ac142f64a-Paper.pdf

[107] J. W. Rae, S. Borgeaud, T. Cai, K. Millican, J. Hoffmann, F. Song, J. Aslanides, S. Henderson, R. Ring, S. Young *et al.*, "Scaling language models: Methods, analysis & insights from training gopher," *arXiv preprint arXiv:2112.11446*, 2021.

[108] A. Chowdhery, S. Narang, J. Devlin, M. Bosma, G. Mishra, A. Roberts, P. Barham, H. W. Chung, C. Sutton, S. Gehrmann *et al.*, "Palm: Scaling language modeling with pathways," *Journal of Machine Learning Research*, vol. 24, no. 240, pp. 1–113, 2023.

[109] T. Le Scao, A. Fan, C. Akiki, E. Pavlick, S. Ilić, D. Hesslow, R. Castagné, A. S. Luccioni, F. Yvon, M. Gallé *et al.*, "Bloom: A 176b-parameter open-access multilingual language model," 2023.

[110] D. E. Rumelhart, G. E. Hinton, and R. J. Williams, "Learning representations by back-propagating errors," *nature*, vol. 323, no. 6088, pp. 533–536, 1986.

[111] T. Kim, J. Oh, N. Kim, S. Cho, and S.-Y. Yun, "Comparing kullback-leibler divergence and mean squared error loss in knowledge distillation," *arXiv preprint arXiv:2105.08919*, 2021.

[112] S. Kato and K. Hotta, "Mse loss with outlying label for imbalanced classification," *arXiv preprint arXiv:2107.02393*, 2021.

[113] J. Ren, M. Zhang, C. Yu, and Z. Liu, "Balanced mse for imbalanced visual regression," in *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, 2022, pp. 7926–7935.

[114] S. Kullback and R. A. Leibler, "On information and sufficiency," *The annals of mathematical statistics*, vol. 22, no. 1, pp. 79–86, 1951.

[115] Z. Zhang and M. Sabuncu, "Generalized cross entropy loss for training deep neural networks with noisy labels," *Advances in neural information processing systems*, vol. 31, 2018.

[116] Y. Ho and S. Wookey, "The real-world-weight cross-entropy loss function: Modeling the costs of mislabeling," *IEEE access*, vol. 8, pp. 4806–4813, 2019.

[117] E. Gordon-Rodriguez, G. Loaiza-Ganem, G. Pleiss, and J. P. Cunningham, "Uses and abuses of the cross-entropy loss: Case studies in modern deep learning," 2020.

[118] A. Mao, M. Mohri, and Y. Zhong, "Cross-entropy loss functions: Theoretical analysis and applications," in *International conference on Machine learning*. PMLR, 2023, pp. 23 803–23 828.

[119] R. Fletcher and M. J. Powell, "A rapidly convergent descent method for minimization," *The computer journal*, vol. 6, no. 2, pp. 163–168, 1963.

[120] Y. LeCun, L. Bottou, Y. Bengio, and P. Haffner, "Gradient-based learning applied to document recognition," *Proceedings of the IEEE*, vol. 86, no. 11, pp. 2278–2324, 1998.

[121] M. Andrychowicz, M. Denil, S. Gomez, M. W. Hoffman, D. Pfau, T. Schaul, B. Shillingford, and N. De Freitas, "Learning to learn by gradient descent by gradient descent," *Advances in neural information processing systems*, vol. 29, 2016.

[122] W. A. Gardner, "Learning characteristics of stochastic-gradient-descent algorithms: A general study, analysis, and critique," *Signal processing*, vol. 6, no. 2, pp. 113–133, 1984.

[123] S.-i. Amari, "Backpropagation and stochastic gradient descent method," *Neurocomputing*, vol. 5, no. 4-5, pp. 185–196, 1993.

[124] L. Eon Bottou, "Online learning and stochastic approximations," *Online learning in neural networks*, vol. 17, no. 9, p. 142, 1998.

[125] A. Krizhevsky, I. Sutskever, and G. E. Hinton, "Imagenet classification with deep convolutional neural networks," *Advances in neural information processing systems*, vol. 25, 2012.

[126] M. Hardt, B. Recht, and Y. Singer, "Train faster, generalize better: Stability of stochastic gradient descent," in *International conference on machine learning*. PMLR, 2016, pp. 1225–1234.

[127] X. Li and F. Orabona, "On the convergence of stochastic gradient descent with adaptive stepsizes," in *The 22nd international conference on artificial intelligence and statistics*. PMLR, 2019, pp. 983–992.

[128] Y. Tian, Y. Zhang, and H. Zhang, "Recent advances in stochastic gradient descent in deep learning," *Mathematics*, vol. 11, no. 3, p. 682, 2023.

[129] D. P. Kingma and J. Ba, "Adam: A method for stochastic optimization," *arXiv preprint arXiv:1412.6980*, 2014.

[130] Z. Zhang, "Improved adam optimizer for deep neural networks," in *2018 IEEE/ACM 26th international symposium on quality of service (IWQoS)*. Ieee, 2018, pp. 1–2.

[131] S. Bock, J. Goppold, and M. Weiß, "An improvement of the convergence proof of the adam-optimizer," *arXiv preprint arXiv:1804.10587*, 2018.

[132] S. Mehta, C. Paunwala, and B. Vaidya, "Cnn based traffic sign classification using adam optimizer," in *2019 international conference on intelligent computing and control systems (ICCS)*. IEEE, 2019, pp. 1293–1298.

[133] D. Yi, J. Ahn, and S. Ji, "An effective optimization method for machine learning based on adam," *Applied Sciences*, vol. 10, no. 3, p. 1073, 2020.

[134] M. M. Lau and K. Hann Lim, "Review of Adaptive Activation Function in Deep Neural Network," in *2018 IEEE-EMBS Conference on Biomedical Engineering and Sciences (IECBES)*, 2018, pp. 686–690.

[135] V. Nair and G. E. Hinton, "Rectified linear units improve restricted boltzmann machines," in *Proceedings of the 27th international conference on machine learning (ICML-10)*, 2010, pp. 807–814.

[136] B. Neyshabur, Y. Wu, R. R. Salakhutdinov, and N. Srebro, "Path-Normalized Optimization of Recurrent Neural Networks with ReLU Activations," in *Advances in Neural Information Processing Systems*, D. Lee, M. Sugiyama, U. Luxburg, I. Guyon, and R. Garnett, Eds., vol. 29. Curran Associates, Inc., 2016. [Online]. Available: https://proceedings.neurips.cc/paper_files/paper/2016/file/74563ba21a90da13dacf2a73e3ddefa7-Paper.pdf

[137] F. Godin, J. Degrave, J. Dambre, and W. De Neve, "Dual Rectified Linear Units (DReLUs): A replacement for tanh activation functions in Quasi-Recurrent Neural Networks," *Pattern Recognition Letters*, vol. 116, pp. 8–14, 2018. [Online]. Available: https://www.sciencedirect.com/science/article/pii/S0167865518305646

[138] S. R. Dubey, S. K. Singh, and B. B. Chaudhuri, "Activation functions in deep learning: A comprehensive survey and benchmark," *Neurocomputing*, vol. 503, pp. 92–108, 2022. [Online]. Available: https://www.sciencedirect.com/science/article/pii/S0925231222008426

[139] K. Cho, B. van Merrienboer, D. Bahdanau, and Y. Bengio, "On the Properties of Neural Machine Translation: Encoder-Decoder Approaches," 2014. [Online]. Available: https://arxiv.org/abs/1409.1259

[140] S. Hochreiter, "The Vanishing Gradient Problem During Learning Recurrent Neural Nets and Problem Solutions," *International Journal of Uncertainty, Fuzziness and Knowledge-Based Systems*, vol. 06, no. 02, pp. 107–116, 1998. [Online]. Available: https://doi.org/10.1142/S0218488598000094

[141] A. Conneau, H. Schwenk, L. Barrault, and Y. Lecun, "Very Deep Convolutional Networks for Text Classification," 2017. [Online]. Available: https://arxiv.org/abs/1606.01781

[142] M.-T. Luong, H. Pham, and C. D. Manning, "Effective approaches to attention-based neural machine translation," 2015. [Online]. Available: https://arxiv.org/abs/1508.04025

[143] D. Bahdanau, K. Cho, and Y. Bengio, "Neural machine translation by jointly learning to align and translate," *arXiv preprint arXiv:1409.0473*, 2014.

[144] M. Wang, S. Lu, D. Zhu, J. Lin, and Z. Wang, "A high-speed and low-complexity architecture for softmax function in deep learning," in *2018 IEEE Asia Pacific Conference on Circuits and Systems (APCCAS)*, 2018, pp. 223–226.

[145] W. Liu, Y. Wen, Z. Yu, and M. Yang, "Large-margin softmax loss for convolutional neural networks," 2017. [Online]. Available: https://arxiv.org/abs/1612.02295

[146] H. Peng, J. Li, Y. Song, and Y. Liu, "Incrementally Learning the Hierarchical Softmax Function for Neural Language Models," *Proceedings of the AAAI Conference on Artificial Intelligence*, vol. 31, no. 1, Feb. 2017. [Online]. Available: https://ojs.aaai.org/index.php/AAAI/article/view/10994

[147] Z. Lin, M. Feng, C. N. d. Santos, M. Yu, B. Xiang, B. Zhou, and Y. Bengio, "A structured self-attentive sentence embedding," *arXiv preprint arXiv:1703.03130*, 2017.

[148] S. Serrano and N. A. Smith, "Is attention interpretable?" *arXiv preprint arXiv:1906.03731*, 2019.

[149] C. A. Córdova Sáenz and K. Becker, "Assessing the use of attention weights to interpret bert-based stance classification," in *IEEE/WIC/ACM International Conference on Web Intelligence and Intelligent Agent Technology*, 2021, pp. 194–201.

[150] S. Wiegreffe and Y. Pinter, "Attention is not not explanation," *arXiv preprint arXiv:1908.04626*, 2019.

[151] P. Shaw, J. Uszkoreit, and A. Vaswani, "Self-attention with relative position representations," 2018. [Online]. Available: https://arxiv.org/abs/1803.02155

[152] H. Zhao, J. Jia, and V. Koltun, "Exploring self-attention for image recognition," in *Proceedings of the IEEE/CVF conference on computer vision and pattern recognition*, 2020, pp. 10 076–10 085.

[153] F. X. Gibbons, "Self-attention and behavior: A review and theoretical update," *Advances in experimental social psychology*, vol. 23, pp. 249–303, 1990.

[154] H. Zhang, I. Goodfellow, D. Metaxas, and A. Odena, "Self-attention generative adversarial networks," in *International conference on machine learning*. PMLR, 2019, pp. 7354–7363.

[155] P. Ramachandran, N. Parmar, A. Vaswani, I. Bello, A. Levskaya, and J. Shlens, "Stand-alone self-attention in vision models," *Advances in neural information processing systems*, vol. 32, 2019.

[156] N. Kitaev, Ł. Kaiser, and A. Levskaya, "Reformer: The efficient transformer," *arXiv preprint arXiv:2001.04451*, 2020.

[157] M. Zaheer, G. Guruganesh, K. A. Dubey, J. Ainslie, C. Alberti, S. Ontanon, P. Pham, A. Ravula, Q. Wang, L. Yang, and A. Ahmed, "Big bird: Transformers for longer sequences," in *Advances in Neural Information Processing Systems*, H. Larochelle, M. Ranzato, R. Hadsell, M. Balcan, and H. Lin, Eds., vol. 33. Curran Associates, Inc., 2020, pp. 17 283–17 297. [Online]. Available: https://proceedings.neurips.cc/paper_files/paper/2020/file/c8512d142a2d849725f31a9a7a361ab9-Paper.pdf

[158] J.-B. Cordonnier, A. Loukas, and M. Jaggi, "On the relationship between self-attention and convolutional layers," *arXiv preprint arXiv:1911.03584*, 2019.

[159] A. Liutkus, O. Cıfka, S.-L. Wu, U. Simsekli, Y.-H. Yang, and G. Richard, "Relative positional encoding for transformers with linear complexity," in *International Conference on Machine Learning*. PMLR, 2021, pp. 7067–7079.

[160] J. Su, M. Ahmed, Y. Lu, S. Pan, W. Bo, and Y. Liu, "Roformer: Enhanced transformer with rotary position embedding," *Neurocomputing*, vol. 568, p. 127063, 2024.

[161] J. Li, X. Wang, Z. Tu, and M. R. Lyu, "On the diversity of multi-head attention," *Neurocomputing*, vol. 454, pp. 14–24, 2021.

[162] J. Li, Z. Tu, B. Yang, M. R. Lyu, and T. Zhang, "Multi-head attention with disagreement regularization," *arXiv preprint arXiv:1810.10183*, 2018.

[163] J. Ainslie, J. Lee-Thorp, M. de Jong, Y. Zemlyanskiy, F. Lebron, and S. Sanghai, "GQA: Training generalized multi-query transformer models from multi-head checkpoints," in *Proceedings of the 2023 Conference on Empirical Methods in Natural Language Processing*, H. Bouamor, J. Pino, and K. Bali, Eds. Singapore: Association for Computational Linguistics, Dec. 2023, pp. 4895–4901. [Online]. Available: https://aclanthology.org/2023.emnlp-main.298/

[164] Z. Khan, M. Khaquan, O. Tafveez, B. Samiwala, and A. A. Raza, "Beyond uniform query distribution: Key-driven grouped query attention," 2024. [Online]. Available: https://arxiv.org/abs/2408.08454

[165] Y. Chen, C. Zhang, X. Gao, R. D. Mullins, G. A. Constantinides, and Y. Zhao, "Optimised grouped-query attention mechanism for transformers," 2024. [Online]. Available: https://arxiv.org/abs/2406.14963

[166] D. Hendrycks and K. Gimpel, "Gaussian Error Linear Units (GELUs)," 2023. [Online]. Available: https://arxiv.org/abs/1606.08415

[167] N. Shazeer, A. Mirhoseini, K. Maziarz, A. Davis, Q. Le, G. Hinton, and J. Dean, "Outrageously large neural networks: The sparsely-gated mixture-of-experts layer," 2017. [Online]. Available: https://arxiv.org/abs/1701.06538

[168] D. Lepikhin, H. Lee, Y. Xu, D. Chen, O. Firat, Y. Huang, M. Krikun, N. Shazeer, and Z. Chen, "Gshard: Scaling giant models with conditional computation and automatic sharding," 2020. [Online]. Available: https://arxiv.org/abs/2006.16668

[169] S. Gupta, S. Mukherjee, K. Subudhi, E. Gonzalez, D. Jose, A. H. Awadallah, and J. Gao, "Sparsely activated mixture-of-experts are robust multi-task learners," *arXiv preprint arXiv:2204.07689*, 2022.

[170] B. Zoph, "Designing effective sparse expert models," in *2022 IEEE International Parallel and Distributed Processing Symposium Workshops (IPDPSW)*. IEEE, 2022, pp. 1044–1044.

[171] S. Rajbhandari, C. Li, Z. Yao, M. Zhang, R. Y. Aminabadi, A. A. Awan, J. Rasley, and Y. He, "DeepSpeed-MoE: Advancing mixture-of-experts inference and training to power next-generation AI scale," in *Proceedings of the 39th International Conference on Machine Learning*, ser. Proceedings of Machine Learning Research, K. Chaudhuri, S. Jegelka, L. Song, C. Szepesvari, G. Niu, and S. Sabato, Eds., vol. 162. PMLR, 17–23 Jul 2022, pp. 18 332–18 346. [Online]. Available: https://proceedings.mlr.press/v162/rajbhandari22a.html

[172] OpenAI, "Models." [Online]. Available: https://platform.openai.com/docs/models

[173] S. Pichai, D. Hassabis, and K. Kavukcuoglu, "Introducing gemini 2.0: our new ai model for the agentic era," 2024. [Online]. Available: https://blog.google/technology/google-deepmind/google-gemini-ai-update-december-2024

[174] G. Team and R. e. a. Anil, "Gemini: A Family of Highly Capable Multimodal Models," 2025. [Online]. Available: https://arxiv.org/abs/2312.11805

[175] Y. Liu, M. Ott, N. Goyal, J. Du, M. Joshi, D. Chen, O. Levy, M. Lewis, L. Zettlemoyer, and V. Stoyanov, "RoBERTa: A Robustly Optimized BERT Pretraining Approach," 2019. [Online]. Available: https://arxiv.org/abs/1907.11692

[176] Z. Yang, Z. Dai, Y. Yang, J. Carbonell, R. R. Salakhutdinov, and Q. V. Le, "XLNet: Generalized Autoregressive Pretraining for Language Understanding," in *Advances in Neural Information Processing Systems*, H. Wallach, H. Larochelle, A. Beygelzimer, F. d'Alché-Buc, E. Fox, and R. Garnett, Eds., vol. 32. Curran Associates, Inc., 2019. [Online]. Available: https://proceedings.neurips.cc/paper_files/paper/2019/file/dc6a7e655d7e5840e66733e9ee67cc69-Paper.pdf

[177] M. Joshi, D. Chen, Y. Liu, D. S. Weld, L. Zettlemoyer, and O. Levy, "Spanbert: Improving pre-training by representing and predicting spans," *Transactions of the Association for Computational Linguistics*, vol. 8, pp. 64–77, 01 2020. [Online]. Available: https://doi.org/10.1162/tacl_a_00300

[178] Z. Lan, M. Chen, S. Goodman, K. Gimpel, P. Sharma, and R. Soricut, "ALBERT: A Lite BERT for Self-supervised Learning of Language Representations," 2020. [Online]. Available: https://arxiv.org/abs/1909.11942

[179] Y. Huang, Y. Cheng, A. Bapna, O. Firat, D. Chen, M. Chen, H. Lee, J. Ngiam, Q. V. Le, Y. Wu, and z. Chen, "GPipe: Efficient Training of Giant Neural Networks using Pipeline Parallelism," in *Advances in Neural Information Processing Systems*, H. Wallach, H. Larochelle, A. Beygelzimer, F. d'Alché-Buc, E. Fox, and R. Garnett, Eds., vol. 32. Curran Associates, Inc., 2019. [Online]. Available: https://proceedings.neurips.cc/paper_files/paper/2019/file/093f65e080a295f8076b1c5722a46aa2-Paper.pdf

[180] Epoch AI, "The Longest Training Run." [Online]. Available: https://epoch.ai/blog/the-longest-training-run

[181] Bigscience, "BigScience Model Training Launched." [Online]. Available: https://bigscience.huggingface.co/blog/model-training-launched

[182] Rao, R., "LLM Training: Mastering the Art of Language Model Development." [Online]. Available: https://www.wevolver.com/article/llm-training-mastering-the-art-of-language-model-development

[183] S. Zhang, S. Roller, N. Goyal, M. Artetxe, M. Chen, S. Chen, C. Dewan, M. Diab, X. Li, X. V. Lin *et al.*, "Opt: Open pre-trained transformer language models," *arXiv preprint arXiv:2205.01068*, 2022.

[184] J. Wei, M. Bosma, V. Y. Zhao, K. Guu, A. W. Yu, B. Lester, N. Du, A. M. Dai, and Q. V. Le, "Finetuned Language Models Are Zero-Shot Learners," 2022. [Online]. Available: https://arxiv.org/abs/2109.01652

[185] T. Kojima, S. S. Gu, M. Reid, Y. Matsuo, and Y. Iwasawa, "Large Language Models are Zero-Shot Reasoners," in *Advances in Neural Information Processing Systems*, S. Koyejo, S. Mohamed, A. Agarwal, D. Belgrave, K. Cho, and A. Oh, Eds., vol. 35. Curran Associates, Inc., 2022, pp. 22 199–22 213. [Online]. Available: https://proceedings.neurips.cc/paper_files/paper/2022/file/8bb0d291acd4acf06ef112099c16f326-Paper-Conference.pdf

[186] A. Radford, J. Wu, R. Child, D. Luan, D. Amodei, I. Sutskever *et al.*, "Language models are unsupervised multitask learners," *OpenAI blog*, vol. 1, no. 8, p. 9, 2019.

[187] C. Hooper, S. Kim, H. Mohammadzadeh, M. W. Mahoney, Y. S. Shao, K. Keutzer, and A. Gholami, "Kvquant: Towards 10 million context length llm inference with kv cache quantization," *Advances in Neural Information Processing Systems*, vol. 37, pp. 1270–1303, 2024.

[188] K. Shuster, S. Poff, M. Chen, D. Kiela, and J. Weston, "Retrieval augmentation reduces hallucination in conversation," 2021. [Online]. Available: https://arxiv.org/abs/2104.07567

[189] O. Ayala and P. Bechard, "Reducing hallucination in structured outputs via retrieval-augmented generation," in *Proceedings of the 2024 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies (Volume 6: Industry Track)*. Association for Computational Linguistics, 2024, p. 228–238. [Online]. Available: http://dx.doi.org/10.18653/v1/2024.naacl-industry.19

[190] B. Sarmah, D. Mehta, B. Hall, R. Rao, S. Patel, and S. Pasquali, "Hybridrag: Integrating knowledge graphs and vector retrieval augmented generation for efficient information extraction," in *Proceedings of the 5th ACM International Conference on AI in Finance*, 2024, pp. 608–616.

[191] Intel Tech, "Optimize Vector Databases, Enhance RAG-Driven Generative AI." [Online]. Available: https://medium.com/intel-tech/optimize-vector-databases-enhance-rag-driven-generative-ai-90c10416cb9c

[192] X. Zhao, X. Zhou, and G. Li, "Chat2data: An interactive data analysis system with rag, vector databases and llms," *Proceedings of the VLDB Endowment*, vol. 17, no. 12, pp. 4481–4484, 2024.

[193] NVIDIA, "What Is Retrieval-Augmented Generation, aka RAG?" [Online]. Available: https://blogs.nvidia.com/blog/what-is-retrieval-augmented-generation/

[194] Y. Zhu, H. Yuan, S. Wang, J. Liu, W. Liu, C. Deng, H. Chen, Z. Liu, Z. Dou, and J.-R. Wen, "Large language models for information retrieval: A survey," *arXiv preprint arXiv:2308.07107*, 2023.

[195] Y. Liu, S. Yavuz, R. Meng, M. Moorthy, S. Joty, C. Xiong, and Y. Zhou, "Exploring the integration strategies of retriever and large language models," *arXiv preprint arXiv:2308.12574*, 2023.

[196] P. Covington, J. Adams, and E. Sargin, "Deep neural networks for youtube recommendations," in *Proceedings of the 10th ACM Conference on Recommender Systems*, ser. RecSys '16. New York, NY, USA: Association for Computing Machinery, 2016, p. 191–198. [Online]. Available: https://doi.org/10.1145/2959100.2959190

[197] H.-T. Cheng, L. Koc, J. Harmsen, T. Shaked, T. Chandra, H. Aradhye, G. Anderson, G. Corrado, W. Chai, M. Ispir, R. Anil, Z. Haque, L. Hong, V. Jain, X. Liu, and H. Shah, "Wide & deep learning for recommender systems," in *Proceedings of the 1st Workshop on Deep Learning for Recommender Systems*, ser. DLRS 2016. New York, NY, USA: Association for Computing Machinery, 2016, p. 7–10. [Online]. Available: https://doi.org/10.1145/2988450.2988454

[198] J. Davidson, B. Liebald, J. Liu, P. Nandy, T. Van Vleet, U. Gargi, S. Gupta, Y. He, M. Lambert, B. Livingston, and D. Sampath, "The youtube video recommendation system," in *Proceedings of the Fourth ACM Conference on Recommender Systems*, ser. RecSys '10. New York, NY, USA: Association for Computing Machinery, 2010, p. 293–296. [Online]. Available: https://doi.org/10.1145/1864708.1864770

[199] C. Burges, T. Shaked, E. Renshaw, A. Lazier, M. Deeds, N. Hamilton, and G. Hullender, "Learning to rank using gradient descent," in *Proceedings of the 22nd International Conference on Machine Learning*, ser. ICML '05. New York, NY, USA: Association for Computing Machinery, 2005, p. 89–96. [Online]. Available: https://doi.org/10.1145/1102351.1102363

[200] Z. Cao, T. Qin, T.-Y. Liu, M.-F. Tsai, and H. Li, "Learning to rank: from pairwise approach to listwise approach," in *Proceedings of the 24th International Conference on Machine Learning*, ser. ICML '07. New York, NY, USA: Association for Computing Machinery, 2007, p. 129–136. [Online]. Available: https://doi.org/10.1145/1273496.1273513

[201] J. Jumper, R. Evans, A. Pritzel, T. Green, M. Figurnov, O. Ronneberger, K. Tunyasuvunakool, R. Bates, A. Žídek, A. Potapenko *et al.*, "Highly accurate protein structure prediction with alphafold," *nature*, vol. 596, no. 7873, pp. 583–589, 2021.

[202] K. He, X. Zhang, S. Ren, and J. Sun, "Deep residual learning for image recognition," in *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, June 2016.

[203] T. Henighan, J. Kaplan, M. Katz, M. Chen, C. Hesse, J. Jackson, H. Jun, T. B. Brown, P. Dhariwal, S. Gray, C. Hallacy, B. Mann, A. Radford, A. Ramesh, N. Ryder, D. M. Ziegler, J. Schulman, D. Amodei, and S. McCandlish, "Scaling laws for autoregressive generative modeling," 2020. [Online]. Available: https://arxiv.org/abs/2010.14701

[204] K. Hong, G. Dai, J. Xu, Q. Mao, X. Li, J. Liu, K. Chen, Y. Dong, and Y. Wang, "Flashdecoding++: Faster large language model inference on gpus," *arXiv preprint arXiv:2311.01282*, 2023.

[205] L. Zhang, M. Wahib, H. Zhang, and S. Matsuoka, "A study of single and multi-device synchronization methods in nvidia gpus," in *2020 IEEE International Parallel and Distributed Processing Symposium (IPDPS)*. IEEE, 2020, pp. 483–493.

[206] J. Dean, G. Corrado, R. Monga, K. Chen, M. Devin, M. Mao, M. Ranzato, A. Senior, P. Tucker, K. Yang *et al.*, "Large scale distributed deep networks," *Advances in neural information processing systems*, vol. 25, 2012.

[207] P. Goyal, P. Dollár, R. Girshick, P. Noordhuis, L. Wesolowski, A. Kyrola, A. Tulloch, Y. Jia, and K. He, "Accurate, large minibatch sgd: Training imagenet in 1 hour," *arXiv preprint arXiv:1706.02677*, 2017.

[208] Z. Chen, L. Shi, X. Liu, J. Li, S. Liu, and Y. Xu, "Osp: Boosting distributed model training with 2-stage synchronization," in *Proceedings of the 52nd International Conference on Parallel Processing*, 2023, pp. 102–111.

[209] Z. Tang, Z. Tang, J. Huang, X. Pan, R. Yan, Y. Wang, A. C. Zhou, S. Shi, X. Chu, and B. Li, "Dreamddp: Accelerating data parallel distributed llm training with layer-wise scheduled partial synchronization," *arXiv preprint arXiv:2502.11058*, 2025.

[210] A. Nabli, L. Fournier, P. Erbacher, L. Serrano, E. Belilovsky, and E. Oyallon, "Acco: Accumulate while you communicate, hiding communications in distributed llm training," *arXiv preprint arXiv:2406.02613*, 2024.

[211] X. Gu, K. Lyu, S. Arora, J. Zhang, and L. Huang, "A quadratic synchronization rule for distributed deep learning," *arXiv preprint arXiv:2310.14423*, 2023.

[212] W. Wen, C. Xu, F. Yan, C. Wu, Y. Wang, Y. Chen, and H. Li, "Terngrad: Ternary gradients to reduce communication in distributed deep learning," *Advances in neural information processing systems*, vol. 30, 2017.

[213] S. Shi, X. Chu, and B. Li, "MG-WFBP: Merging gradients wisely for efficient communication in distributed deep learning," *IEEE Transactions on Parallel and Distributed Systems*, vol. 32, no. 8, pp. 1903–1917, 2021.

[214] Z. Jia, M. Zaharia, and A. Aiken, "Beyond data and model parallelism for deep neural networks." in *Proceedings of Machine Learning and Systems*, A. Talwalkar, V. Smith, and M. Zaharia, Eds., vol. 1, 2019, pp. 1–13. [Online]. Available: https://proceedings.mlsys.org/paper_files/paper/2019/file/b422680f3db0986ddd7f8f126baaf0fa-Paper.pdf

[215] B. Wang, Q. Xu, Z. Bian, and Y. You, "Tesseract: Parallelize the tensor parallelism efficiently," in *Proceedings of the 51st International Conference on Parallel Processing*, 2022, pp. 1–11.

[216] K. Osawa, S. Li, and T. Hoefler, "PipeFisher: Efficient Training of Large Language Models Using Pipelining and Fisher Information Matrices," in *Proceedings of Machine Learning and Systems*, D. Song, M. Carbin, and T. Chen, Eds., vol. 5. Curan, 2023, pp. 708–727. [Online]. Available: https://proceedings.mlsys.org/paper_files/paper/2023/file/dd064459e9ef4100671ba326f0f96f2b-Paper-mlsys2023.pdf

[217] L. Song, X. Qian, H. Li, and Y. Chen, "Pipelayer: A pipelined reram-based accelerator for deep learning," in *2017 IEEE international symposium on high performance computer architecture (HPCA)*. IEEE, 2017, pp. 541–552.

[218] C. He, S. Li, M. Soltanolkotabi, and S. Avestimehr, "Pipetransformer: Automated elastic pipelining for distributed training of transformers," *arXiv preprint arXiv:2102.03161*, 2021.

[219] M. P. Forum, "Mpi: A message-passing interface standard," 1994.

[220] R. Thakur, R. Rabenseifner, and W. Gropp, "Optimization of collective communication operations in mpich," *The International Journal of High Performance Computing Applications*, vol. 19, no. 1, pp. 49–66, 2005. [Online]. Available: https://doi.org/10.1177/1094342005051521

[221] J. Fei, C.-Y. Ho, A. N. Sahu, M. Canini, and A. Sapio, "Efficient sparse collective communication and its application to accelerate distributed deep learning," in *Proceedings of the 2021 ACM SIGCOMM 2021 Conference*, ser. SIGCOMM '21. New York, NY, USA: Association for Computing Machinery, 2021, p. 676–691. [Online]. Available: https://doi.org/10.1145/3452296.3472904

[222] G. Xu, Z. Le, Y. Chen, Z. Lin, Z. Jin, Y. Miao, and C. Li, "AutoCCL: Automated Collective Communication Tuning for Accelerating Distributed and Parallel DNN Training," in *22nd USENIX Symposium on Networked Systems Design and Implementation (NSDI 25)*. Philadelphia, PA: USENIX Association, Apr. 2025, pp. 667–683. [Online]. Available: https://www.usenix.org/conference/nsdi25/presentation/xu-guanbin

[223] M. Zhai, J. He, Z. Ma, Z. Zong, R. Zhang, and J. Zhai, "SmartMoE: Efficiently training Sparsely-Activated models through combining offline and online parallelization," in *2023 USENIX Annual Technical Conference (USENIX ATC 23)*. Boston, MA: USENIX Association, Jul. 2023, pp. 961–975. [Online]. Available: https://www.usenix.org/conference/atc23/presentation/zhai

[224] H. Huang, N. Ardalani, A. Sun, L. Ke, H.-H. S. Lee, A. Sridhar, S. Bhosale, C.-J. Wu, and B. Lee, "Towards moe deployment: Mitigating inefficiencies in mixture-of-expert (moe) inference," *arXiv preprint arXiv:2303.06182*, 2023.

[225] Mitra, T. and Borkar, R. and Elmeleegy, A. and Kapasi, U. and Darvish, B., "How NVIDIA GB200 NVL72 and NVIDIA Dynamo Boost Inference Performance for MoE Models." [Online]. Available: https://developer.nvidia.com/blog/how-nvidia-gb200-nvl72-and-nvidia-dynamo-boost-inference-performance-for-moe-models/

[226] Elmeleegy, A. and Raj, S. and Slechta, B. and Mehta, V., "Demystifying AI Inference Deployments for Trillion Parameter Large Language Models." [Online]. Available: https://developer.nvidia.com/blog/demystifying-ai-inference-deployments-for-trillion-parameter-large-language-models/

[227] B. Wu, Y. Zhong, Z. Zhang, S. Liu, F. Liu, Y. Sun, G. Huang, X. Liu, and X. Jin, "Fast Distributed Inference Serving for Large Language Models," 2024. [Online]. Available: https://arxiv.org/abs/2305.05920

[228] Z. Zheng, X. Ren, F. Xue, Y. Luo, X. Jiang, and Y. You, "Response Length Perception and Sequence Scheduling: An LLM-Empowered LLM Inference Pipeline," in *Advances in Neural Information Processing Systems*, A. Oh, T. Naumann, A. Globerson, K. Saenko, M. Hardt, and S. Levine, Eds., vol. 36. Curran Associates, Inc., 2023, pp. 65 517–65 530. [Online]. Available: https://proceedings.neurips.cc/paper_files/paper/2023/file/ce7ff3405c782f761fac7f849b41ae9a-Paper-Conference.pdf

[229] R. Li, D. Fu, C. Shi, Z. Huang, and G. Lu, "Efficient LLMs Training and Inference: An Introduction," *IEEE Access*, vol. 13, pp. 32 944–32 970, 2025.

[230] R. Y. Aminabadi, S. Rajbhandari, A. A. Awan, C. Li, D. Li, E. Zheng, O. Ruwase, S. Smith, M. Zhang, J. Rasley *et al.*, "Deepspeed-inference: enabling efficient inference of transformer models at unprecedented scale," in *SC22: International Conference for High Performance Computing, Networking, Storage and Analysis*. IEEE, 2022, pp. 1–15.

[231] P. Chaturvedi, A. Khan, M. Tian, E. Huerta, and H. Zheng, "Inference-optimized ai and high performance computing for gravitational wave detection at scale," *Frontiers in Artificial Intelligence*, vol. 5, p. 828672, 2022.

[232] Verma, S. and Vaidya, N., "Mastering LLM Techniques: Inference Optimization." [Online]. Available: https://developer.nvidia.com/blog/mastering-llm-techniques-inference-optimization/

[233] P. Patel, E. Choukse, C. Zhang, A. Shah, Goiri, S. Maleki, and R. Bianchini, "Splitwise: Efficient Generative LLM Inference Using Phase Splitting," in *2024 ACM/IEEE 51st Annual International Symposium on Computer Architecture (ISCA)*, 2024, pp. 118–132.

[234] V. Karpukhin, B. Oguz, S. Min, P. S. Lewis, L. Wu, S. Edunov, D. Chen, and W.-t. Yih, "Dense passage retrieval for open-domain question answering." in *EMNLP (1)*, 2020, pp. 6769–6781.

[235] A. Mansurova, A. Mansurova, and A. Nugumanova, "QA-RAG: Exploring LLM reliance on external knowledge," *Big Data and Cognitive Computing*, vol. 8, no. 9, p. 115, 2024.

[236] Y. Gao, Y. Xiong, X. Gao, K. Jia, J. Pan, Y. Bi, Y. Dai, J. Sun, H. Wang, and H. Wang, "Retrieval-augmented generation for large language models: A survey," *arXiv preprint arXiv:2312.10997*, vol. 2, no. 1, 2023.

[237] D. Quinn, M. Nouri, N. Patel, J. Salihu, A. Salemi, S. Lee, H. Zamani, and M. Alian, "Accelerating Retrieval-Augmented Generation," in *Proceedings of the 30th ACM International Conference on Architectural Support for Programming Languages and Operating Systems, Volume 1*, ser. ASPLOS '25. New York, NY, USA: Association for Computing Machinery, 2025, p. 15–32. [Online]. Available: https://doi.org/10.1145/3669940.3707264

[238] J. F. Shoch and J. A. Hupp, "Measured performance of an Ethernet local network," *Commun. ACM*, vol. 23, no. 12, p. 711–721, Dec. 1980. [Online]. Available: https://doi.org/10.1145/359038.359044

[239] B. Lowekamp, D. O'Hallaron, and T. Gross, "Topology discovery for large ethernet networks," in *Proceedings of the 2001 Conference on Applications, Technologies, Architectures, and Protocols for Computer Communications*, ser. SIGCOMM '01. New York, NY, USA: Association for Computing Machinery, 2001, p. 237–248. [Online]. Available: https://doi.org/10.1145/383059.383078

[240] B. Stephens, A. Cox, W. Felter, C. Dixon, and J. Carter, "PAST: scalable ethernet for data centers," in *Proceedings of the 8th International Conference on Emerging Networking Experiments and Technologies*, ser. CoNEXT '12. New York, NY, USA: Association for Computing Machinery, 2012, p. 49–60. [Online]. Available: https://doi.org/10.1145/2413176.2413183

[241] T. Hoefler, T. Schneider, and A. Lumsdaine, "Optimized Routing for Large-Scale InfiniBand Networks," in *2009 17th IEEE Symposium on High Performance Interconnects*, 2009, pp. 103–111.

[242] K. Hintze, S. Graham, S. Dunlap, and P. Sweeney, "InfiniBand Network Monitoring: Challenges and Possibilities," in *Critical Infrastructure Protection XV*, J. Staggs and S. Shenoi, Eds. Cham: Springer International Publishing, 2022, pp. 187–208.

[243] R. Buyya, T. Cortes, and H. Jin, *An Introduction to the InfiniBand Architecture*, 2002, pp. 616–632.

[244] Z. Wang, L. Luo, Q. Ning, C. Zeng, W. Li, X. Wan, P. Xie, T. Feng, K. Cheng, X. Geng, T. Wang, W. Ling, K. Huo, P. An, K. Ji, S. Zhang, B. Xu, R. Feng, T. Ding, K. Chen, and C. Guo, "SRNIC: A scalable architecture for RDMA NICs," in *20th USENIX Symposium on Networked Systems Design and Implementation (NSDI 23)*. Boston, MA: USENIX Association, Apr. 2023, pp. 1–14. [Online]. Available: https://www.usenix.org/conference/nsdi23/presentation/wang-zilong

[245] F. Daoud, A. Watad, and M. Silberstein, "GPUrdma: GPU-side library for high performance networking from GPU kernels," in *Proceedings of the 6th International Workshop on Runtime and Operating Systems for Supercomputers*, ser. ROSS '16. New York, NY, USA: Association for Computing Machinery, 2016. [Online]. Available: https://doi.org/10.1145/2931088.2931091

[246] E. Agostini, D. Rossetti, and S. Potluri, "GPUDirect Async: Exploring GPU synchronous communication techniques for InfiniBand clusters," *Journal of Parallel and Distributed Computing*, vol. 114, pp. 28–45, 2018. [Online]. Available: https://www.sciencedirect.com/science/article/pii/S0743731517303386

[247] A. Greenberg, J. R. Hamilton, N. Jain, S. Kandula, C. Kim, P. Lahiri, D. A. Maltz, P. Patel, and S. Sengupta, "VL2: a scalable and flexible data center network," *SIGCOMM Comput. Commun. Rev.*, vol. 39, no. 4, p. 51–62, Aug. 2009. [Online]. Available: https://doi.org/10.1145/1594977.1592576

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

[249] A. Singla, C.-Y. Hong, L. Popa, and P. B. Godfrey, "Jellyfish: Networking data centers randomly," in *9th USENIX Symposium on Networked Systems Design and Implementation (NSDI 12)*, 2012, pp. 225–238.

[250] C. Tang, "Meta's Hyperscale Infrastructure: Overview and Insights," *Commun. ACM*, vol. 68, no. 2, p. 52–63, Jan. 2025. [Online]. Available: https://doi.org/10.1145/3701296

[251] A. Andreyev, W. Xu, and A. Eckert, "Reinventing Facebook's data center network," 2019. [Online]. Available: https://engineering.fb.com/2019/03/14/data-center-engineering/f16-minipack/

[252] NVIDIA, "NVIDIA Blackwell Architecture." [Online]. Available: https://www.nvidia.com/en-us/data-center/technologies/blackwell-architecture/

[253] Micron, "HBM3E." [Online]. Available: https://www.micron.com/products/memory/hbm/hbm3e

[254] NVIDIA, "NVIDIA NVLink-C2C." [Online]. Available: https://www.nvidia.com/en-us/data-center/nvlink-c2c/

[255] NVIDIA, "NVIDIA Blackwell Architecture Technical Brief," 2024. [Online]. Available: https://resources.nvidia.com/en-us-blackwell-architecture

[256] NVIDIA, "NVIDIA Grace." [Online]. Available: https://www.nvidia.com/en-us/data-center/grace-cpu/

[257] Y. Wei, Y. C. Huang, H. Tang, N. Sankaran, I. Chadha, D. Dai, O. Oluwole, V. Balan, and E. Lee, "9.3 nvlink-c2c: A coherent off package chip-to-chip interconnect with 40gbps/pin single-ended signaling," in *2023 IEEE International Solid-State Circuits Conference (ISSCC)*. IEEE, 2023, pp. 160–162.

[258] I. Burstein, "Nvidia Data Center Processing Unit (DPU) Architecture," in *2021 IEEE Hot Chips 33 Symposium (HCS)*, 2021, pp. 1–20.

[259] F. Inc., "An introduction to nvidia connectx-5 network adapter," 2024. [Online]. Available: https://www.fs.com/blog/an-introduction-to-nvidia-connectx5-network-adapter-2558.html

[260] N. Shankarappa, "Accelerating with xdp over mellanox connectx nics," 2020. [Online]. Available: https://developer.nvidia.com/blog/accelerating-with-xdp-over-mellanox-connectx-nics/

[261] A. Elmeleegy, "NVIDIA Contributes NVIDIA GB200 NVL72 Designs to Open Compute Project," 2024. [Online]. Available: https://developer.nvidia.com/blog/nvidia-contributes-nvidia-gb200-nvl72-designs-to-open-compute-project/

[262] NVIDIA, "NVIDIA NVLink and NVLink Switch." [Online]. Available: https://www.nvidia.com/en-us/data-center/nvlink/

[263] Brian Slechta and Nick Comly and Ashraf Eassa and Joe DeLaere and Shivam Raj, "NVIDIA NVLink and NVIDIA NVSwitch Supercharge Large Language Model Inference," 2024. [Online]. Available: https://developer.nvidia.com/blog/nvidia-nvlink-and-nvidia-nvswitch-supercharge-large-language-model-inference/

[264] A. Singla, P. B. Godfrey, and A. Kolla, "High throughput data center topology design," in *11th USENIX Symposium on Networked Systems Design and Implementation (NSDI 14)*, 2014, pp. 29–41.

[265] F. Inc., "What is an aggregate switch?" 2023. [Online]. Available: https://www.fs.com/blog/what-is-an-aggregate-switch-1340.html

[266] Y. Li, D. Wei, X. Chen, Z. Song, R. Wu, Y. Li, X. Jin, and W. Xu, "Dumbnet: a smart data center network fabric with dumb switches," in *Proceedings of the Thirteenth EuroSys Conference*, ser. EuroSys '18. New York, NY, USA: Association for Computing Machinery, 2018. [Online]. Available: https://doi.org/10.1145/3190508.3190531

[267] Cisco, "Whitepaper: Cisco aci multi-tier architecture," 2024. [Online]. Available: https://www.cisco.com/c/en/us/solutions/collateral/data-center-virtualization/application-centric-infrastructure/white-paper-c11-742214.pdf

[268] HPE, "What is spine-leaf architecture?" 2024. [Online]. Available: https://www.hpe.com/emea_africa/en/what-is/spine-leaf-architecture.html

[269] M. Alizadeh and T. Edsall, "On the data path performance of leaf-spine datacenter fabrics," in *2013 IEEE 21st Annual Symposium on High-Performance Interconnects*, 2013, pp. 71–74.

[270] NVIDIA, "QM87xx 1U HDR 200Gb/s InfiniBand Switch Systems User Manual." [Online]. Available: https://docs.nvidia.com/qm87xx-1u-hdr-200gb-s-infiniband-switch-systems-user-manual.pdf

[271] NVIDIA, "NVIDIA Quantum InfiniBand Switches." [Online]. Available: https://www.nvidia.com/en-us/networking/infiniband-switching/

[272] NVIDIA, "NVIDIA Quantum-2 InfiniBand Platform Datasheet." [Online]. Available: https://nvdam.widen.net/s/dps8txlsrf/infiniband-ndr-400g-architecture-datasheet-1620877-r4

[273] NVIDIA, "NVIDIA Spectrum-X Datasheet." [Online]. Available: https://resources.nvidia.com/en-us-networking-ai/networking-ethernet-1

[274] N. Rasmussen and W. Torell, "Data center projects: establishing a floor plan," *White Paper*, vol. 144, 2007.

[275] Vertiv, "Understanding Coolant Distribution Units (CDUs) for Liquid Cooling," 2023. [Online]. Available: https://www.vertiv.com/en-us/about/news-and-insights/articles/educational-articles/understanding-coolant-distribution-units-cdus-for-liquid-cooling/

[276] BOYD, "Data Center Cooling Systems: Coolant Distribution Unit Liquid Cooling." [Online]. Available: https://www.boydcorp.com/blog/data-center-cooling-systems-coolant-distribution-unit-liquid-cooling.html

[277] S. Kala and S. Mills, "DC Power Distribution Unit for V2 Open Rack," 2015. [Online]. Available: https://www.opencompute.org/wiki/Open_Rack/SpecsAndDesigns

[278] H. Keyhani, "Open Rack V3 48V PSU Specification Rev 1.0," 2022. [Online]. Available: https://www.opencompute.org/wiki/Open_Rack/SpecsAndDesigns

[279] Meta, "Meta Open Rack V3 BBU Module," 2022. [Online]. Available: https://www.opencompute.org/wiki/Open_Rack/SpecsAndDesigns

[280] N. Blach, M. Besta, D. De Sensi, J. Domke, H. Harake, S. Li, P. Iff, M. Konieczny, K. Lakhotia, A. Kubicek *et al.*, "A {High-Performance} design, implementation, deployment, and evaluation of the slim fly network," in *21st USENIX Symposium on Networked Systems Design and Implementation (NSDI 24)*, 2024, pp. 1025–1044.

[281] J. Liu, W. Jiang, P. Wyckoff, D. K. Panda, D. Ashton, D. Buntinas, W. Gropp, and B. Toonen, "Design and implementation of mpich2 over infiniband with rdma support," in *18th International Parallel and Distributed Processing Symposium, 2004. Proceedings.* IEEE, 2004, p. 16.

[282] D. De Sensi, L. Pichetti, F. Vella, T. De Matteis, Z. Ren, L. Fusco, M. Turisini, D. Cesarini, K. Lust, A. Trivedi *et al.*, "Exploring gpu-to-gpu communication: Insights into supercomputer interconnects," in *SC24: International Conference for High Performance Computing, Networking, Storage and Analysis*. IEEE, 2024, pp. 1–15.

[283] B. Lebiednik, A. Mangal, and N. Tiwari, "A survey and evaluation of data center network topologies," *arXiv preprint arXiv:1605.01701*, 2016.

[284] R. Niranjan Mysore, A. Pamboris, N. Farrington, N. Huang, P. Miri, S. Radhakrishnan, V. Subramanya, and A. Vahdat, "Portland: a scalable fault-tolerant layer 2 data center network fabric," *SIGCOMM Comput. Commun. Rev.*, vol. 39, no. 4, p. 39–50, Aug. 2009. [Online]. Available: https://doi.org/10.1145/1594977.1592575

[285] H. Xu, C. Feng, and B. Li, "Temperature aware workload management in geo-distributed datacenters," *SIGMETRICS Perform. Eval. Rev.*, vol. 41, no. 1, p. 373–374, Jun. 2013. [Online]. Available: https://doi.org/10.1145/2494232.2465539

[286] Baxtel, "Global data center map." [Online]. Available: https://baxtel.com/map

[287] AWS, "Global infrastructure regions & azs." [Online]. Available: https://aws.amazon.com/about-aws/global-infrastructure/regions_az/

[288] Microsoft, "Global infrastructure." [Online]. Available: https://azure.microsoft.com/en-us/explore/global-infrastructure

[289] Microsoft, "What are azure availability zones?" [Online]. Available: https://learn.microsoft.com/en-us/azure/reliability/availability-zones-overview?tabs=azure-cli

[290] Meta, "Global (asia, europe, u.s.) - meta data centers." [Online]. Available: https://datacenters.atmeta.com/all-locations/

[291] Google, "Global locations - regions & zones." [Online]. Available: https://cloud.google.com/about/locations

[292] J. Romero, J. Yin, N. Laanait, B. Xie, M. T. Young, S. Treichler, V. Starchenko, A. Borisevich, A. Sergeev, and M. Matheson, "Accelerating collective communication in data parallel training across deep learning frameworks," in *19th USENIX Symposium on Networked Systems Design and Implementation (NSDI 22)*, 2022, pp. 1027–1040.

[293] S. Shi, X. Chu, and B. Li, "Exploiting simultaneous communications to accelerate data parallel distributed deep learning," in *IEEE INFOCOM 2021-IEEE Conference on Computer Communications*. IEEE, 2021, pp. 1–10.

[294] J. Fang, H. Fu, G. Yang, and C.-J. Hsieh, "Redsync: reducing synchronization bandwidth for distributed deep learning training system," *Journal of Parallel and Distributed Computing*, vol. 133, pp. 30–39, 2019.

[295] N. Xie, T. Norman, D. Grewe, and D. Vytiniotis, "Synthesizing optimal parallelism placement and reduction strategies on hierarchical systems for deep learning," *Proceedings of Machine Learning and Systems*, vol. 4, pp. 548–566, 2022.

[296] F. Seide, H. Fu, J. Droppo, G. Li, and D. Yu, "1-bit stochastic gradient descent and its application to data-parallel distributed training of speech dnns." in *Interspeech*, vol. 2014. Singapore, 2014, pp. 1058–1062.

[297] N. Alnaasan, A. Jain, A. Shafi, H. Subramoni, and D. K. Panda, "Accdp: Accelerated data-parallel distributed dnn training for modern gpu-based hpc clusters," in *2022 IEEE 29th International Conference on High Performance Computing, Data, and Analytics (HiPC)*, 2022, pp. 32–41.

[298] M. Cho, U. Finkler, D. Kung, and H. Hunter, "BlueConnect: Decomposing All-Reduce for Deep Learning on Heterogeneous Network Hierarchy," in *Proceedings of Machine Learning and Systems*, A. Talwalkar, V. Smith, and M. Zaharia, Eds., vol. 1, 2019, pp. 241–251. [Online]. Available: https://proceedings.mlsys.org/paper_files/paper/2019/file/0c8abcf158ed12d0dd94480681186fda-Paper.pdf

[299] S. Legtchenko, H. Williams, K. Razavi, A. Donnelly, R. Black, A. Douglas, N. Cheriere, D. Fryer, K. Mast, A. D. Brown, A. Klimovic, A. Slowey, and A. Rowstron, "Understanding Rack-Scale Disaggregated Storage," in *9th USENIX Workshop on Hot Topics in Storage and File Systems (HotStorage 17)*. Santa Clara, CA: USENIX Association, Jul. 2017. [Online]. Available: https://www.usenix.org/conference/hotstorage17/program/presentation/legtchenko

[300] K. Lim, J. Chang, T. Mudge, P. Ranganathan, S. K. Reinhardt, and T. F. Wenisch, "Disaggregated memory for expansion and sharing in blade servers," in *Proceedings of the 36th Annual International Symposium on Computer Architecture*, ser. ISCA '09. New York, NY, USA: Association for Computing Machinery, 2009, p. 267–278. [Online]. Available: https://doi.org/10.1145/1555754.1555789

[301] K. Lim, Y. Turner, J. R. Santos, A. AuYoung, J. Chang, P. Ranganathan, and T. F. Wenisch, "System-level implications of disaggregated memory," in *IEEE International Symposium on High-Performance Comp Architecture*, 2012, pp. 1–12.

[302] S. Han, N. Egi, A. Panda, S. Ratnasamy, G. Shi, and S. Shenker, "Network support for resource disaggregation in next-generation datacenters," in *Proceedings of the Twelfth ACM Workshop on Hot Topics in Networks*, ser. HotNets-XII. New York, NY, USA: Association for Computing Machinery, 2013. [Online]. Available: https://doi.org/10.1145/2535771.2535778

[303] P. X. Gao, A. Narayan, S. Karandikar, J. Carreira, S. Han, R. Agarwal, S. Ratnasamy, and S. Shenker, "Network requirements for resource disaggregation," in *12th USENIX Symposium on Operating Systems Design and Implementation (OSDI 16)*. Savannah, GA: USENIX Association, Nov. 2016, pp. 249–264. [Online]. Available: https://www.usenix.org/conference/osdi16/technical-sessions/presentation/gao

[304] J. Kundu, W. Guo, A. BanaGozar, U. De Alwis, S. Sengupta, P. Gupta, and A. Mallik, "Performance modeling and workload analysis of distributed large language model training and inference," in *2024 IEEE International Symposium on Workload Characterization (IISWC)*. IEEE, 2024, pp. 57–67.

[305] Q. Hu, Z. Ye, Z. Wang, G. Wang, M. Zhang, Q. Chen, P. Sun, D. Lin, X. Wang, Y. Luo *et al.*, "Characterization of large language model development in the datacenter," in *21st USENIX Symposium on Networked Systems Design and Implementation (NSDI 24)*, 2024, pp. 709–729.

[306] M. Sponner, B. Waschneck, and A. Kumar, "Ai-driven performance modeling for ai inference workloads," *Electronics*, vol. 11, no. 15, p. 2316, 2022.

[307] W. Jiang, S. Subramanian, C. Graves, G. Alonso, A. Yazdanbakhsh, and V. Dadu, "Rago: Systematic performance optimization for retrieval-augmented generation serving," *arXiv preprint arXiv:2503.14649*, 2025.

[308] B. Sharma, L. Yang, and H. Pham, "Multi-Tenancy for AI Inference at Meta Scale," 2023. [Online]. Available: https://atscaleconference.com/multi-tenancy-for-ai-inference-at-meta-scale/

[309] Meta, "Llama3 model," 2024. [Online]. Available: https://github.com/meta-llama/llama3

[310] Meta, "Llama 2: Open foundation and fine-tuned chat models." [Online]. Available: https://huggingface.co/meta-llama

[311] Mistral, "Mistral-7b and mixtral models." [Online]. Available: https://github.com/mistralai/mistral-inference

[312] G. Liu, H. Yin, B. Zhu, J. Chen, C.-W. Ngo, and Y.-G. Jiang, "Retrieval augmented recipe generation," in *2025 IEEE/CVF Winter Conference on Applications of Computer Vision (WACV)*. IEEE, 2025, pp. 2453–2463.

[313] F. Liang, Z. Zhang, H. Lu, V. Leung, Y. Guo, and X. Hu, "Communication-efficient large-scale distributed deep learning: A comprehensive survey," *arXiv preprint arXiv:2404.06114*, 2024.

[314] W. Xiao, S. Ren, Y. Li, Y. Zhang, P. Hou, Z. Li, Y. Feng, W. Lin, and Y. Jia, "{AntMan}: Dynamic scaling on {GPU} clusters for deep learning," in *14th USENIX Symposium on Operating Systems Design and Implementation (OSDI 20)*, 2020, pp. 533–548.

[315] A. Gholami, Z. Yao, S. Kim, C. Hooper, M. W. Mahoney, and K. Keutzer, "AI and Memory Wall," *IEEE Micro*, vol. 44, no. 3, pp. 33–39, 2024.

[316] J. Gu, Y. Lee, Y. Zhang, M. Chowdhury, and K. G. Shin, "Efficient memory disaggregation with infiniswap," in *14th USENIX Symposium on Networked Systems Design and Implementation (NSDI 17)*, 2017, pp. 649–667.

[317] P. Zuo, J. Sun, L. Yang, S. Zhang, and Y. Hua, "One-sided {RDMA-Conscious} extendible hashing for disaggregated memory," in *2021 USENIX Annual Technical Conference (USENIX ATC 21)*, 2021, pp. 15–29.

[318] I. Calciu, M. T. Imran, I. Puddu, S. Kashyap, H. A. Maruf, O. Mutlu, and A. Kolli, "Rethinking software runtimes for disaggregated memory," in *Proceedings of the 26th ACM International Conference on Architectural Support for Programming Languages and Operating Systems*, 2021, pp. 79–92.

[319] Z. Wang, X. Wei, J. Gu, H. Xie, R. Chen, and H. Chen, "{ODRP}:{On-Demand} remote paging with programmable {RDMA}," in *22nd USENIX Symposium on Networked Systems Design and Implementation (NSDI 25)*, 2025, pp. 1101–1115.

[320] S. Han, N. Egi, A. Panda, S. Ratnasamy, G. Shi, and S. Shenker, "Network support for resource disaggregation in next-generation datacenters," in *Proceedings of the Twelfth ACM Workshop on Hot Topics in Networks*, 2013, pp. 1–7.

[321] SNIA, "Enterprise and Data Center SSD Form Factor - the end of the 2.5in disk era?" [Online]. Available: https://www.snia.org/sites/default/files/SSSI/EDSFF%20Webcast%208-4-2020%20fnl.pdf

[322] SNIA, "The Latest on Form Factors." [Online]. Available: https://snia.org/sites/default/files/SSSI/CMSS24/CMSS24-Hands-The-Latest-on-Form-Factors.pdf

[323] Ogle, M. and Lynn, B. and Armstrong, J., "Dell Technologies Focuses on Standardizing EDSFF Form Factor for Future Servers." [Online]. Available: https://infohub.delltechnologies.com/en-us/p/dell-technologies-focuses-on-standardizing-edsff-form-factor-for-future-servers/

[324] Intel, "Intel® Data Center SSDs based on EDSFF*; the perfect fit." [Online]. Available: https://www.intel.de/content/www/de/de/products/docs/memory-storage/solid-state-drives/edsff-brief.html

[325] B. Jacob, S. W. Ng, and D. T. Wang, "Chapter 10 - dram memory system organization," in *Memory Systems*, B. Jacob, S. W. Ng, and D. T. Wang, Eds. San Francisco: Morgan Kaufmann, 2008, pp. 409–424. [Online]. Available: https://www.sciencedirect.com/science/article/pii/B9780123797513500126

[326] J. Schneider and I. Smalley, "What is a dual in-line memory module (dimm)?" 2024. [Online]. Available: https://www.ibm.com/think/topics/dimm

[327] J. Kanade, "What is a dual in-line memory module (dimm)? meaning, characteristics, and types," 2023. [Online]. Available: https://www.spiceworks.com/tech/tech-general/articles/what-is-dimm/

[328] D. M. Harris and S. L. Harris, "8 - memory and i/o systems," in *Digital Design and Computer Architecture (Second Edition)*, second edition ed., D. M. Harris and S. L. Harris, Eds. Boston: Morgan Kaufmann, 2013, pp. 474–580. [Online]. Available: https://www.sciencedirect.com/science/article/pii/B9780123944245000082

[329] X. Wang, J. Wang, X. Tao, M. Yang, and J. Lai, "Design and implementation of ddr3 sdram controller," in *2018 14th IEEE International Conference on Solid-State and Integrated Circuit Technology (ICSICT)*. IEEE, 2018, pp. 1–3.

[330] U. Kang, H.-J. Chung, S. Heo, D.-H. Park, H. Lee, J. H. Kim, S.-H. Ahn, S.-H. Cha, J. Ahn, D. Kwon *et al.*, "8 gb 3-d ddr3 dram using through-silicon-via technology," *IEEE Journal of Solid-State Circuits*, vol. 45, no. 1, pp. 111–119, 2009.

[331] Samsung, "Ddr3 | dram | samsung semiconductor global." [Online]. Available: https://semiconductor.samsung.com/dram/ddr/ddr3/

[332] M. A. Islam, M. Y. Arafath, and M. J. Hasan, "Design of ddr4 sdram controller," in *8th International Conference on Electrical and Computer Engineering*, 2014, pp. 148–151.

[333] S. Lee, H. Cho, Y. H. Son, Y. Ro, N. S. Kim, and J. H. Ahn, "Leveraging power-performance relationship of energy-efficient modern dram devices," *IEEE Access*, vol. 6, pp. 31 387–31 398, 2018.

[334] Micron Technology, Inc., "Tn-40-40: Ddr4 point-to-point design guide." [Online]. Available: https://www.mouser.com/pdfDocs/Micron_DDR4_Design_Guide.pdf?srsltid=AfmBOorGptc-D9pZ_YIZMJiEJxCf6Aq83LKLWoVmlwFafNuPmxixFins

[335] M. H. Hajkazemi, M. K. Tavana, and H. Homayoun, "Wide i/o or lpddr? exploration and analysis of performance, power and temperature trade-offs of emerging dram technologies in embedded mpsocs," in *2015 33rd IEEE International Conference on Computer Design (ICCD)*, 2015, pp. 62–69.

[336] K.-S. Ha, C.-K. Lee, D. Lee, D. Moon, H.-R. Hwang, D. Park, Y.-H. Kim, Y. H. Son, B. Na, S. Lee, Y.-S. Park, H.-J. Kwon, T.-Y. Oh, Y.-S. Sohn, S.-J. Bae, K.-I. Park, and J.-B. Lee, "A 7.5 gb/s/pin 8-gb lpddr5 sdram with various high-speed and low-power techniques," *IEEE Journal of Solid-State Circuits*, vol. 55, no. 1, pp. 157–166, 2020.

[337] Samsung, "Lpddr | dram | samsung semiconductor global." [Online]. Available: https://semiconductor.samsung.com/dram/lpddr/

[338] C. Clos, "A study of non-blocking switching networks," *Bell System Technical Journal*, vol. 32, no. 2, pp. 406–424, 1953.

[339] A. Singh, J. Ong, A. Agarwal, G. Anderson, A. Armistead, R. Bannon, S. Boving, G. Desai, B. Felderman, P. Germano *et al.*, "Jupiter rising: A decade of clos topologies and centralized control in google's datacenter network," *ACM SIGCOMM computer communication review*, vol. 45, no. 4, pp. 183–197, 2015.

[340] H. Choo, S.-M. Yoo, and H. Y. Youn, "Processor scheduling and allocation for 3d torus multicomputer systems," *IEEE Transactions on Parallel and Distributed Systems*, vol. 11, no. 5, pp. 475–484, 2000.

[341] P. López and J. Duato, "Deadlock-free adaptive routing algorithms for the 3d-torus: Limitations and solutions," in *International Conference on Parallel Architectures and Languages Europe*. Springer, 1993, pp. 684–687.

[342] P. Costa, A. Donnelly, G. O'Shea, and A. Rowstron, "Camcubeos: a key-based network stack for 3d torus cluster topologies," in *Proceedings of the 22nd international symposium on High-performance parallel and distributed computing*, 2013, pp. 73–84.

[343] S. Cheng, W. Zhong, K. E. Isaacs, and K. Mueller, "Visualizing the topology and data traffic of multi-dimensional torus interconnect networks," *IEEE Access*, vol. 6, pp. 57 191–57 204, 2018.

[344] J. Kim, W. J. Dally, S. Scott, and D. Abts, "Technology-driven, highly-scalable dragonfly topology," *ACM SIGARCH Computer Architecture News*, vol. 36, no. 3, pp. 77–88, 2008.

[345] J. Kim, W. Dally, S. Scott, and D. Abts, "Cost-efficient dragonfly topology for large-scale systems," *IEEE micro*, vol. 29, no. 1, pp. 33–40, 2009.

[346] A. Shpiner, Z. Haramaty, S. Eliad, V. Zdornov, B. Gafni, and E. Zahavi, "Dragonfly+: Low cost topology for scaling datacenters," in *2017 IEEE 3rd International Workshop on High-Performance Interconnection Networks in the Exascale and Big-Data Era (HiPINEB)*. IEEE, 2017, pp. 1–8.

[347] Vortex, "Vortex: OpenCL Compatible RISC-V GPGPU." [Online]. Available: https://vortex.cc.gatech.edu/

[348] Vortex, "Vortex Tutorial at MICRO 2022." [Online]. Available: https://vortex.cc.gatech.edu/micro2022?

[349] B. Tine, K. P. Yalamarthy, F. Elsabbagh, and K. Hyesoon, "Vortex: Extending the risc-v isa for gpgpu and 3d-graphics," in *MICRO-54: 54th Annual IEEE/ACM International Symposium on Microarchitecture*, ser. MICRO '21. New York, NY, USA: Association for Computing Machinery, 2021, p. 754–766. [Online]. Available: https://doi.org/10.1145/3466752.3480128

[350] Hyesoon Kim et al., "Vortex Workshop and Tutorials at MICRO-57 (3rd November 2024)." [Online]. Available: https://github.com/vortexgpgpu/vortex_tutorials

[351] Asanović, K. et al., "Rocket Chip (Berkeley)." [Online]. Available: https://github.com/chipsalliance/rocket-chip

[352] Luchterhandt, L. et al., "Towards a Rocket Chip Based Implementation of the RISC-V GPC Architecture." [Online]. Available: https://chipyard.readthedocs.io/en/latest/

[353] C. Lameter, "Numa (non-uniform memory access): An overview: Numa becomes more common because memory controllers get close to execution units on microprocessors." *Queue*, vol. 11, no. 7, pp. 40–51, 2013.

[354] N. Denoyelle, B. Goglin, A. Ilic, E. Jeannot, and L. Sousa, "Modeling non-uniform memory access on large compute nodes with the cache-aware roofline model," *IEEE Transactions on Parallel and Distributed Systems*, vol. 30, no. 6, pp. 1374–1389, 2018.

[355] Z. Majo and T. R. Gross, "(mis) understanding the numa memory system performance of multithreaded workloads," in *2013 IEEE International Symposium on Workload Characterization (IISWC)*. IEEE, 2013, pp. 11–22.

[356] A. Radford, J. W. Kim, C. Hallacy, A. Ramesh, G. Goh, S. Agarwal, G. Sastry, A. Askell, P. Mishkin, J. Clark *et al.*, "Learning transferable visual models from natural language supervision," in *International conference on machine learning*. PmLR, 2021, pp. 8748–8763.

[357] H. Shi, M. Hayat, Y. Wu, and J. Cai, "Proposalclip: Unsupervised open-category object proposal generation via exploiting clip cues," in *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, 2022, pp. 9611–9620.

[358] Y. Ma, G. Xu, X. Sun, M. Yan, J. Zhang, and R. Ji, "X-clip: End-to-end multi-grained contrastive learning for video-text retrieval," in *Proceedings of the 30th ACM international conference on multimedia*, 2022, pp. 638–647.

[359] C. Mavromatis and G. Karypis, "Gnn-rag: Graph neural retrieval for large language model reasoning," *arXiv preprint arXiv:2405.20139*, 2024.

[360] Y. Hu, Z. Lei, Z. Zhang, B. Pan, C. Ling, and L. Zhao, "Grag: Graph retrieval-augmented generation," *arXiv preprint arXiv:2405.16506*, 2024.

[361] C. Mavromatis and G. Karypis, "Gnn-rag: Graph neural retrieval for large language model reasoning," *arXiv preprint arXiv:2405.20139*, 2024.

[362] J. Jang, H. Choi, H. Bae, S. Lee, M. Kwon, and M. Jung, "{CXL-ANNS}:{Software-Hardware} collaborative memory disaggregation and computation for {Billion-Scale} approximate nearest neighbor search," in *2023 USENIX Annual Technical Conference (USENIX ATC 23)*, 2023, pp. 585–600.

[363] P. Ristoski and H. Paulheim, "Rdf2vec: Rdf graph embeddings for data mining," in *The Semantic Web–ISWC 2016: 15th International Semantic Web Conference, Kobe, Japan, October 17–21, 2016, Proceedings, Part I 15*. Springer, 2016, pp. 498–514.

[364] M. Cochez, P. Ristoski, S. P. Ponzetto, and H. Paulheim, "Global rdf vector space embeddings," in *The Semantic Web–ISWC 2017: 16th International Semantic Web Conference, Vienna, Austria, October 21–25, 2017, Proceedings, Part I 16*. Springer, 2017, pp. 190–207.

[365] P. Ristoski and H. Paulheim, "Rdf2vec: Rdf graph embeddings for data mining," in *The Semantic Web–ISWC 2016: 15th International Semantic Web Conference, Kobe, Japan, October 17–21, 2016, Proceedings, Part I 15*. Springer, 2016, pp. 498–514.

[366] M. Cochez, P. Ristoski, S. P. Ponzetto, and H. Paulheim, "Global rdf vector space embeddings," in *The Semantic Web–ISWC 2017: 16th International Semantic Web Conference, Vienna, Austria, October 21–25, 2017, Proceedings, Part I 16*. Springer, 2017, pp. 190–207.

[367] Z. Wu, S. Pan, F. Chen, G. Long, C. Zhang, and P. S. Yu, "A comprehensive survey on graph neural networks," *IEEE transactions on neural networks and learning systems*, vol. 32, no. 1, pp. 4–24, 2020.

[368] F. Scarselli, M. Gori, A. C. Tsoi, M. Hagenbuchner, and G. Monfardini, "The graph neural network model," *IEEE transactions on neural networks*, vol. 20, no. 1, pp. 61–80, 2008.

[369] J. Zhou, G. Cui, S. Hu, Z. Zhang, C. Yang, Z. Liu, L. Wang, C. Li, and M. Sun, "Graph neural networks: A review of methods and applications," *AI open*, vol. 1, pp. 57–81, 2020.

[370] Y. Malkov, A. Ponomarenko, A. Logvinov, and V. Krylov, "Approximate nearest neighbor algorithm based on navigable small world graphs," *Information Systems*, vol. 45, pp. 61–68, 2014.

[371] Y. A. Malkov and D. A. Yashunin, "Efficient and robust approximate nearest neighbor search using hierarchical navigable small world graphs," *IEEE transactions on pattern analysis and machine intelligence*, vol. 42, no. 4, pp. 824–836, 2018.

[372] X. Xu, C. Li, Y. Wang, and Y. Xia, "Multiattribute approximate nearest neighbor search based on navigable small world graph," *Concurrency and Computation: Practice and Experience*, vol. 32, no. 24, p. e5970, 2020.

[373] M. Douze, A. Guzhva, C. Deng, J. Johnson, G. Szilvasy, P.-E. Mazaré, M. Lomeli, L. Hosseini, and H. Jégou, "The faiss library," *arXiv preprint arXiv:2401.08281*, 2024.

[374] D. Danopoulos, C. Kachris, and D. Soudris, "Approximate similarity search with faiss framework using fpgas on the cloud," in *International Conference on Embedded Computer Systems*. Springer, 2019, pp. 373–386.

[375] MLcommons, "DLRM Workload for Recommendation:." [Online]. Available: https://github.com/mlcommons/algorithmic-efficiency/issues/32

[376] Meta, "Deep Learning Recommendation Model for Personalization and Recommendation Systems:." [Online]. Available: https://github.com/facebookresearch/dlrm

[377] S. Liu, N. Zheng, H. Kang, X. Simmons, J. Zhang, M. Langer, W. Zhu, M. Lee, and Z. Wang, "Embedding optimization for training large-scale deep learning recommendation systems with embark," in *Proceedings of the 18th ACM Conference on Recommender Systems*, 2024, pp. 622–632.

[378] M. Naumov, D. Mudigere, H.-J. M. Shi, J. Huang, N. Sundaraman, J. Park, X. Wang, U. Gupta, C.-J. Wu, A. G. Azzolini, D. Dzhulgakov, A. Mallevich, I. Cherniavskii, Y. Lu, R. Krishnamoorthi, A. Yu, V. Kondratenko, S. Pereira, X. Chen, W. Chen, V. Rao, B. Jia, L. Xiong, and M. Smelyanskiy, "Deep learning recommendation model for personalization and recommendation systems," 2019. [Online]. Available: https://arxiv.org/abs/1906.00091

[379] G. Bosilca, T. Herault, A. Rezmerita, and J. Dongarra, "On scalability for mpi runtime systems," in *2011 IEEE International Conference on Cluster Computing*. IEEE, 2011, pp. 187–195.

[380] G. M. Shipman, T. S. Woodall, R. L. Graham, A. B. Maccabe, and P. G. Bridges, "Infiniband scalability in Open MPI," in *Proceedings 20th IEEE International Parallel & Distributed Processing Symposium*. IEEE, 2006, pp. 10–pp.

[381] R. Zambre, M. Grodowitz, A. Chandramowlishwaran, and P. Shamis, "Breaking band: A breakdown of high-performance communication," in *Proceedings of the 48th International Conference on Parallel Processing*, 2019, pp. 1–10.

[382] J. Huang, K. Ouyang, Y. Zhai, J. Liu, M. Si, K. Raffenetti, H. Zhou, A. Hori, Z. Chen, Y. Guo *et al.*, "Accelerating mpi collectives with process-in-process-based multi-object techniques," in *Proceedings of the 32nd International Symposium on High-Performance Parallel and Distributed Computing*, 2023, pp. 333–334.

[383] J.-L. Vay, A. Almgren, J. Bell, L. Ge, D. Grote, M. Hogan, O. Kononenko, R. Lehe, A. Myers, C. Ng, J. Park, R. Ryne, O. Shapoval, M. Thévenet, and W. Zhang, "Warp-X: A new exascale computing platform for beam–plasma simulations," *Nuclear Instruments and Methods in Physics Research Section A: Accelerators, Spectrometers, Detectors and Associated Equipment*, vol. 909, pp. 476–479, 2018, 3rd European Advanced Accelerator Concepts workshop (EAAC2017). [Online]. Available: https://www.sciencedirect.com/science/article/pii/S0168900218300524

[384] P. C. Liewer and V. K. Decyk, "A general concurrent algorithm for plasma particle-in-cell simulation codes," *Journal of Computational Physics*, vol. 85, no. 2, pp. 302–322, 1989. [Online]. Available: https://www.sciencedirect.com/science/article/pii/0021999189901538

[385] UCLA Plasma Simulation Group, "PIC Skeleton Codes," 2017. [Online]. Available: https://github.com/UCLA-Plasma-Simulation-Group/PIC-skeleton-codes

[386] P. R. Spalart and V. Venkatakrishnan, "On the role and challenges of cfd in the aerospace industry," *The Aeronautical Journal*, vol. 120, no. 1223, pp. 209–232, 2016.

[387] OpenFOAM, "OpenFOAM v12," 2024. [Online]. Available: https://openfoam.org/

[388] C. Petersen, "Building the Case for UALink™: A Dedicated Scale-Up Memory Semantic Fabric," 2024. [Online]. Available: https://www.asteralabs.com/building-the-case-for-ualink-a-dedicated-scale-up-memory-semantic-fabric/

[389] UALink Consortium, "Introducing ualink 200g 1.0 specification," 2025. [Online]. Available: https://ualinkconsortium.org/wp-content/uploads/2025/04/UALink-1.0-White_Paper_v3.pdf

[390] E. Chan, M. Heimlich, A. Purkayastha, and R. Van De Geijn, "Collective communication: theory, practice, and experience," *Concurrency and Computation: Practice and Experience*, vol. 19, no. 13, pp. 1749–1783, 2007.

[391] J. Bruck, C.-T. Ho, S. Kipnis, and D. Weathersby, "Efficient algorithms for all-to-all communications in multi-port message-passing systems," in *Proceedings of the Sixth Annual ACM Symposium on Parallel Algorithms and Architectures*, ser. SPAA '94. New York, NY, USA: Association for Computing Machinery, 1994, p. 298–309. [Online]. Available: https://doi.org/10.1145/181014.181756

[392] C. Lutz, S. Breß, S. Zeuch, T. Rabl, and V. Markl, "Pump up the volume: Processing large data on gpus with fast interconnects," in *Proceedings of the 2020 ACM SIGMOD International Conference on Management of Data*, 2020, pp. 1633–1649.

[393] Y. Zhang, R. Nazaraliyev, S. B. Dutta, A. Marquez, K. Barker, and N. Abu-Ghazaleh, "Nvbleed: Covert and side-channel attacks on nvidia multi-gpu interconnect," *arXiv preprint arXiv:2503.17847*, 2025.

[394] Habana labs. [Online]. Available: https://habana.ai/

[395] Graphcore. [Online]. Available: https://www.graphcore.ai/

[396] Y. Chen, T. Luo, S. Liu, S. Zhang, L. He, J. Wang, L. Li, T. Chen, Z. Xu, N. Sun *et al.*, "Dadiannao: A machine-learning supercomputer," in *2014 47th Annual IEEE/ACM International Symposium on Microarchitecture*. IEEE, 2014, pp. 609–622.

[397] J.-W. Jang, S. Lee, D. Kim, H. Park, A. S. Ardestani, Y. Choi, C. Kim, Y. Kim, H. Yu, H. Abdel-Aziz *et al.*, "Sparsity-aware and re-configurable npu architecture for samsung flagship mobile soc," in *2021 ACM/IEEE 48th Annual International Symposium on Computer Architecture (ISCA)*. IEEE, 2021, pp. 15–28.

[398] Meta, "MTIA," 2024. [Online]. Available: https://ai.meta.com/blog/next-generation-meta-training-inference-accelerator-AI-MTIA/

[399] Amazon, "Trainium," 2024. [Online]. Available: https://aws.amazon.com/ko/ai/machine-learning/trainium/

[400] Amazon, "Inferentia," 2024. [Online]. Available: https://aws.amazon.com/ko/ai/machine-learning/inferentia/

[401] Microsoft, "Maia," 2024. [Online]. Available: https://azure.microsoft.com/en-us/blog/azure-maia-for-the-era-of-ai-from-silicon-to-software-to-systems/

[402] Intel, "Gaudi," 2024. [Online]. Available: https://www.intel.com/content/www/us/en/products/details/processors/ai-accelerators/gaudi.html

[403] NVIDIA Newsroom, "NVIDIA Unveils NVLink Fusion for Industry to Build Semi-Custom AI Infrastructure With NVIDIA Partner Ecosystem." [Online]. Available: https://nvidianews.nvidia.com/news/nvidia-nvlink-fusion-semi-custom-ai-infrastructure-partner-ecosystem

[404] UCIe Consortium, "UCIe 2.0 specification," 2024. [Online]. Available: https://www.uciexpress.org/2-0-spec-download

**Notices & Disclaimers**

Panmnesia, the Panmnesia logo, and other Panmnesia marks are trademarks of Panmnesia, Inc. or its subsidiaries. Other names and brands may be claimed as the property of others.
All content contained in this document is protected by applicable copyright laws. Any unauthorized use, reproduction, distribution, or transmission of the content is strictly prohibited without prior written consent of Panmnesia.
All information included herein is provided "AS IS." Panmnesia hereby disclaims all warranties, representations, and guarantees of any kind with respect to the information in this document, including without limitation, warranties of merchantability, non-infringement, accuracy, completeness, timeliness, or fitness for any particular purpose.
Panmnesia reserves the right to make corrections, modifications, enhancements, improvements, and any other changes to this document, at any time without notice.

Neither Panmnesia nor any of its affiliates, officers, employees, or representatives shall bear any responsibility or liability whatsoever for any errors, omissions, or consequences arising from the use of or reliance upon any information included herein. Any recipient should conduct their own due diligence before making any decisions based on this information.

**© 2025 Panmnesia, Inc. All rights reserved.**