Your network doesn't end at the NIC

A case for unifying the inter-host and intra-host networks in (AI) datacenters

Raj Joshi
Harvard University

Saksham Agarwal
UIUC

ChonLam Lao
Harvard University

Minlan Yu
Harvard University

Abstract

Modern ML workloads increasingly rely on direct communication between host devices—such as GPUs, NVMe SSDs, and DRAM—spanning intra-host and inter-host networks. However, today's intra-host network lacks hardware-level primitives for routing across heterogeneous interconnects, hindering efficient use of alternative paths and leading to suboptimal performance under failures or congestion. Furthermore, the inter-host network treats the NIC as the endpoint, with intra-host interconnects like PCIe running oblivious to inter-host network protocols. This prevents leveraging multiple paths for communication between host devices across different servers. To address these limitations, we propose expanding the datacenter network layer to encompass the intra-host network, making intra-host devices first-class network endpoints. Our scheme envisions hardware-level routing and forwarding across multiple intra-host interconnects and makes intra-host devices visible to the inter-host network. This unified approach provides a principled foundation for robust, efficient peer-to-peer communication between storage and compute hardware devices in AI datacenters.

CCS Concepts

• **Networks** → **Network architectures**; **Data center networks**.

Keywords

AI Datacenters, Intra-host Network, GPU Networking, PCIe

HotNets '25, College Park, MD, USA

ACM ISBN 979-8-4007-2280-6/25/11
https://doi.org/10.1145/3772356.3772415

ACM Reference Format:
Raj Joshi, Saksham Agarwal, ChonLam Lao, Minlan Yu. 2025. Your network doesn't end at the NIC: A case for unifying the inter-host and intra-host networks in (AI) datacenters. In *The 24th ACM Workshop on Hot Topics in Networks (HotNets '25), November 17–18, 2025, College Park, MD, USA.* ACM, New York, NY, USA, 9 pages. https://doi.org/10.1145/3772356.3772415

1 Introduction

The network layer in computer networks is primarily responsible for delivering data from a source host to a destination host across interconnected networks. The Internet Protocol (IP), the standard in both the Internet and datacenter environments, specifies how data is addressed and routed to its destination. It also enables efficient use of path diversity by supporting routing around link failures [32] and load balancing across equal-cost paths [16].

Network layer protocols have traditionally assumed that the network begins and ends at the host's network interface card (NIC). Data transfers between the NIC and application buffers are handled by hardware-level protocols of intra-host interconnects like PCIe [2, 60], which operate independently of the inter-host Ethernet/IP protocols. This design makes sense for classical networking applications, which typically run on the host CPU and only move data between the NIC and the CPU memory (DRAM). Servers for these workloads typically feature one or more NICs connected to the CPU via single-rooted PCIe topologies [42], ensuring a single, deterministic path between NIC and DRAM, and therefore not requiring any dynamic intra-host routing.

The rise of machine learning (ML) workloads such as large language model (LLM) training and inference has driven the adoption of servers with multiple accelerators (GPUs), storage devices (NVMe SSDs), and RDMA NICs (RNICs), leading to larger and more complex intra-host network topologies. Figure 1 illustrates the common intra-host topology used in server designs from major vendors [4, 9, 14, 29, 35, 55], with similar designs adopted by multi-GPU cloud VMs [6, 23, 50]. In these architectures, each CPU socket connects to multiple PCIe switches, with each switch linking a GPU, an RNIC, and several NVMe SSDs. The PCIe interconnect enables

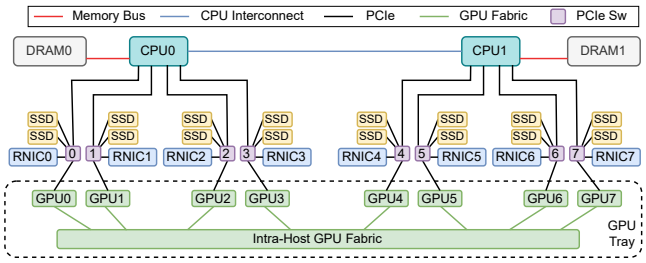

Figure 1: Intra-host network in a multi-GPU server.

data transfers across devices (e.g., RNIC↔GPU, GPU↔SSD) and to DRAM, while GPUs are also connected via a higher-bandwidth GPU fabric for fast inter-GPU transfers. Although server architectures and intra-host topologies have evolved in response to new application demands, intra-host transfers still rely on rudimentary hardware protocols originally designed for simple, single-path device-to-DRAM transfers.

The hardware protocols for intra-host data transfers have two key limitations. First, they lack native support for routing and forwarding data across heterogeneous interconnects. For instance, in Figure 1, while the hardware supports transfers within the PCIe fabric between any pair of devices (RNIC, SSD, GPU or DRAM) and within the GPU fabric between any pair of GPUs, it offers no hardware-level primitives for transfers spanning *both* PCIe and GPU fabrics. This limits efficient use of alternative paths. For example, if `RNIC0` encounters PCIe link failure or congestion while transferring data to `DRAM0` via `PCIeSw0`, it could instead use an alternative path through `GPU0`→`GPU1`→`PCIeSw1`. However, existing hardware primitives limit the use of the latter, leading to suboptimal performance and resource utilization under failures or congestion. While alternative paths can be utilized through application-level primitives, it introduces significant overheads (see §2.2).

The second limitation is that intra-host hardware protocols operate independently of inter-host network protocols, offering the inter-host network no visibility into the rich "last-mile" topology within the server. Servers statically link each GPU to its nearest local RNIC, so when a remote GPU sends data to `GPU0` in Figure 1, it is routed to `RNIC0`. If there is failure or congestion at the switch-to-`RNIC0` link, inter-host protocols remain unaware of alternative intra-host paths to `GPU0` via `RNICs1-7`, preventing their effective use. Such seemingly local failures can stall entire distributed training jobs, resulting in minutes of wasted computation and leaving expensive accelerators idle (§2.3).

In this paper, we propose expanding the datacenter network layer's scope beyond the host NIC to include the intra-host network, making intra-host devices first-class network endpoints. We envision hardware-level routing and forwarding across multiple intra-host interconnects (e.g., PCIe and GPU fabric) and making intra-host devices and intra-host link states visible to the inter-host network. Our blueprint to realize this vision includes: (i) a global addressing scheme for all intra-host devices, (ii) unified routing tables across intra- and inter-host networks, and (iii) network hardware-level forwarding across PCIe and GPU fabric interconnects.

Table 1: ML driving device-to-device communication

Device Pairs	ML Workload Tasks
GPU↔GPU	LLM training: collective communication; LLM serving: tensor, pipeline, expert parallelism; prefill-decode disaggregation
GPU↔DRAM	LLM training: optimizer state & model parameter offload, model checkpointing, input loading; LLM serving: KV cache offload, compute offload
GPU↔SSD	LLM training: data ingestion, checkpointing; LLM serving: model loading, KV cache offload; DLRM inference
RNIC↔GPU,SSD,DRAM	All above when devices are in different servers

The rest of the paper analyzes current device-to-device communication and its limitations (§2), describes our proposed blueprint (§3), and discusses future directions (§4) and related work (§5).

2 Inefficiencies in Data Transfers for ML Workloads

In this section, we first examine how modern ML workloads drive device-to-device transfers within and across datacenter hosts (§2.1). We then show how the lack of routing and forwarding across heterogeneous host interconnects causes intra-host inefficiencies (§2.2), and how the independent operation of intra-host and inter-host networks leads to inefficient data transfers across hosts (§2.3).

2.1 Data Transfers in ML Workloads

Modern ML workloads have driven the rise of data transfers between various host devices—such as CPU-attached DRAM, GPUs, RNICs, and SSDs—across the intra-host network. Table 1 summarizes the key device pairs and the ML tasks that drive the inter-device communication. The dominant pattern is GPU↔GPU communication, driven by collective operations in LLM training [21, 34], and by parallelisms [25, 51, 59] as well as prefill-decode disaggregation [41, 65] in LLM serving. Intra-server GPU communication uses GPU fabric, while inter-server communication uses RNIC pairs via GPUDirect RDMA [40]. GPU↔DRAM communication is common for KV cache offload [25] and can also target remote DRAM using RDMA [45]. Other GPU↔DRAM communication includes optimizer/model parameter offload [47], in-memory checkpointing [61], and input data loading [3] in LLM training. GPU↔SSD communication, enabled by GPUDirect Storage [37], is increasingly used for model checkpointing and data ingestion in training [33, 39], fast model loading and KV

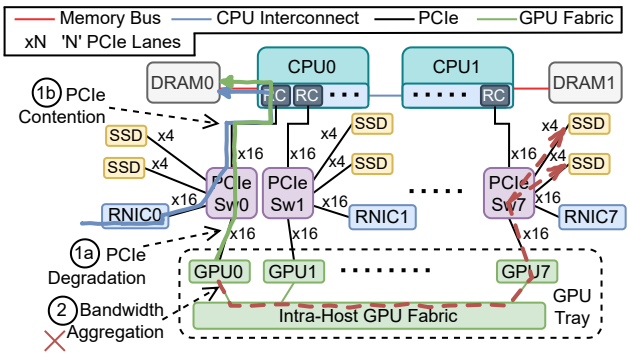

Figure 2: Intra-host communication faces degradation, contention and lacks bandwidth aggregation.

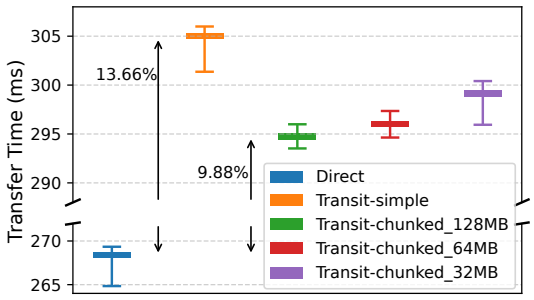

Figure 3: Transfer time for 6.4 GB KV cache from `GPU0` to `DRAM0` directly (PCIe) vs. app-driven transit via `GPU1`. Whiskers: min, 25th, 50th,75th, and 99th percentiles.

cache offload in inference [48, 64], and large-scale embedding lookups in DLRM [63].

While Table 1 shows pair-wise device communication, many transfers happen concurrently in practice. For instance, during LLM training, a GPU may save a checkpoint to DRAM while also sending gradients to a remote GPU [61]. Emerging systems like RAG [30], mixture-of-experts [8], and prefill-decode disaggregation [45] are making host device communication patterns more complex and dynamic.

2.2 Inefficient Intra-host Data Transfers

We now illustrate intra-host data transfer inefficiencies using KV cache offloading as a case study. Before examining two example scenarios, we briefly overview KV cache offloading and its datapath.

In today's LLM serving (inference) systems, the key-value (KV) cache is a large intermediate state which is used to predict the next output token [57]. While serving multiple requests, the KV cache for all the requests cannot simultaneously fit into the GPU's limited onboard memory. As a result, today's state-of-the-art LLM serving systems offload the KV cache to the local CPU-attached DRAM [26, 28], SSDs [48, 52], and other GPU's memory [58]. In a distributed setup, GPUs can also offload KV cache to the DRAM and SSDs in a remote server [45].

KV cache offloading datapath. We consider `GPU0` offloading KV cache to `DRAM0` in Figure 2. Each host device, including DRAM, exposes a memory region that is assigned a unique address range within a unified physical address space by the BIOS/UEFI. To initiate the transfer, the GPU's DMA engine is given the destination address and transfer size. It then issues multiple Memory-Write TLPs on its PCIe interface, each with a destination memory address belonging to `DRAM0`.[1] `PCIeSw0` then uses static address-based routing to forward these TLPs upstream to the PCIe root complex (RC) in `CPU0`, which converts them into cacheline writes and sends the data to `DRAM0` over the memory bus.

Other than the CPU-attached memory, `GPU0` can also offload KV cache to the SSDs using the GPUDirect Storage (GDS) [36] paradigm. GDS typically involves two steps. First, the SSD driver on the CPU creates an *NVMe Write* entry in the SSD's submission queue in host DRAM. This entry contains pointers that reference the KV cache pages in the GPU memory. The CPU then rings the SSD's doorbell via a PCIe Memory-Write TLP, prompting the NVMe controller to fetch and execute the command, thereby pulling data directly from the GPU using Memory-Read TLPs and committing it to its flash [46]. Modern CPUs can forward Memory Read/Write PCIe TLPs to any PCIe RC in the host, allowing `GPU0` to offload KV cache to any desired set of SSDs.

Scenario 1: Existing data transfer mechanisms can't efficiently route around failed or contended PCIe link. Failures of a subset of PCIe lanes (resulting in PCIe bandwidth degradation) or even complete PCIe link failures are common in multi-GPU servers deployed at scale [10, 31]. If the PCIe link connecting `GPU0` to `PCIeSw0` (see ⑴a in Figure 2) suffers bandwidth degradation, it naturally affects the performance of KV cache offloading from `GPU0` to `DRAM0`. Similarly, the KV cache transfer from `GPU0` to `DRAM0` can face contention (see ⑴b in Figure 2) if there is a concurrent KV cache transfer from a remote GPU to `DRAM0` via `RNIC0`. Notice that `GPU0` has alternative paths to reach `DRAM0`, such as via `GPU1`, but the current intra-host network lacks primitives for network-layer re-routing over these paths. As a result, only application-layer re-routing is possible through the following steps: (i) selecting an alternative path (e.g., via `GPU1`), (ii) allocating memory on `GPU1` as a staging buffer, and (iii) transferring the KV cache to `GPU1` over the GPU fabric, then from `GPU1` to `DRAM0` via PCIe DMA (as described above).

However, application-driven forwarding has several drawbacks. Figure 3 shows that the simplest application-driven approach ("Transit-simple"), where `GPU0` first transfers the

[1] For simplicity, we assume IOMMU is disabled; otherwise, address translation is required [1].

entire 6.4 GB KV cache to GPU1 before GPU1 sends it to DRAM0 takes ~13.66% longer than the direct path.[2] This is primarily because the second transfer cannot start until the first one finishes. This method uses minimal control logic, but requires a large staging buffer. Alternatively, using a smaller staging buffer reduces memory usage by dividing the transfer into pipelined chunks, allowing parallel transfers from GPU0 to GPU1 and GPU1 to DRAM0. However, each chunk requires application intervention, increasing control overhead (see "Transit-chunked" in Figure 3) and resulting in at least ~9.88% longer transfer times. In both approaches, staging buffers consume valuable memory on the transit GPU and require complex mechanisms to avoid interfering with the GPU's local workloads [49].

Scenario 2: Existing data transfer mechanisms can't utilize the total available GPU↔SSD bandwidth. When GPU0 is offloading its KV cache to the SSDs, it can reach all 16 SSDs within the server using its PCIe interface, thanks to the aforementioned CPU support for TLP forwarding. However, since GPU0's PCIe interface has x16 lanes and each SSD is accessible via x4 lanes, GPU0 can simultaneously offload KV cache to only 4 SSDs. Notice that GPU0's fabric interface offers additional, higher-bandwidth to access more SSDs simultaneously – e.g., the NVIDIA H100's GPU fabric provides 3.6 Tbps versus 512 Gbps on PCIe [44]. However, the current intra-host network lacks network-level primitives to simultaneously leverage the GPU fabric bandwidth for KV cache offloading. As in Scenario 1, the only way for GPU0 to aggregate bandwidth across interfaces is via the application layer, which must transfer KV cache segments to staging buffers on intermediate GPUs before offloading to different SSDs.

2.3 Inefficient Inter-host Data Transfers

In this subsection, we use distributed LLM training as a case study to illustrate an example scenario of inefficient data transfer across the inter-host network.

Modern LLM training systems operate in distributed environments with tens of thousands of GPUs [13, 21], using collective communication to synchronize parameters, aggregate gradients, and coordinate updates. We focus on a single data transfer within a collective operation, where say GPU_A0 in server A sends data to GPU_B0 in server B. When devices on different servers communicate, the intra-host network provides last-mile connectivity, with RNICs serving as gateways between intra- and inter-host networks.

Datapath for transfer across GPUs in different servers. First, on each server, GPU_A0 (or GPU_B0) allocates a buffer and registers it with its nearest RNIC (RNIC_A0 or RNIC_B0). Next, RNIC_A0 and RNIC_B0 establish an RDMA Queue Pair (QP) by exchanging their network coordinates (e.g., GIDs, QP numbers) and memory handles (address and remote key). Server A then posts an RDMA Write work request to RNIC_A0. RNIC_A0 pulls data from GPU_A0's buffer by issuing PCIe Memory-Read TLPs and receiving PCIe Completion Data TLPs. This data is sent to RNIC_B0 using RoCEv2 over the inter-host network, which in turn pushes it into GPU_B0's designated buffer via PCIe Memory-Write TLPs. GPU collective communication libraries such as NCCL are built on top of this basic data movement primitive, and other devices follow a similar workflow for inter-host communication.

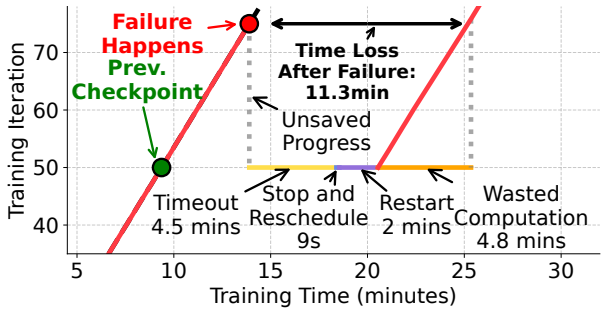

Figure 4: Training with failure

Scenario 3: Existing data transfer mechanisms can't route around failed RNICs. Failures are common in LLM training, causing significant downtime and GPU underutilization [10, 11]. Network-related failures, including faults in switch-to-RNIC links, RNICs, or RNIC-PCIe connections make up over 10% of all reported failures in distributed LLM training [10]. To quantify the impact of a single RNIC failure, we trained GPT2-20B on a 4-machine, 32×A100 GPU cluster (Megatron-LM, TP=2, PP=4, DP=4). Figure 4 depicts the training progress when RNIC_B0 fails and GPU_B0 become unreachable (although remains healthy). Training immediately stalls and waits for a default timeout (4.5 mins[3]) before rescheduling. After this timeout, the training is stopped and rescheduled on a new healthy server, replacing server B – a standard practice due to the tightly-coupled, synchronous nature of distributed training [20, 56]. In our setup, rescheduling took 9 secs, but this step can take up to 15 mins at production scale (e.g., 8192 GPUs) [27]. After that, restarting requires reloading checkpoints which took 2 mins, and this overhead can increase at larger scales [61]. Finally, after restart, all unsaved iterations since the last checkpoint must be rerun; with 10-second iterations and 25 lost iterations, this resulted in ~4.8 mins of wasted computation.

In summary, a *single* RNIC failure resulted in ~11.3 minutes of downtime and GPU underutilization in our setup; and the impact is even greater at larger scales. The core issue is the

[2] Conducted on a PCIe4.0 multi-GPU server with NVIDIA A100 GPUs.

[3] A minute-level timeout is set to avoid mistakenly terminating training due to transient network fluctuations or short-term anomalies at large scale [10].

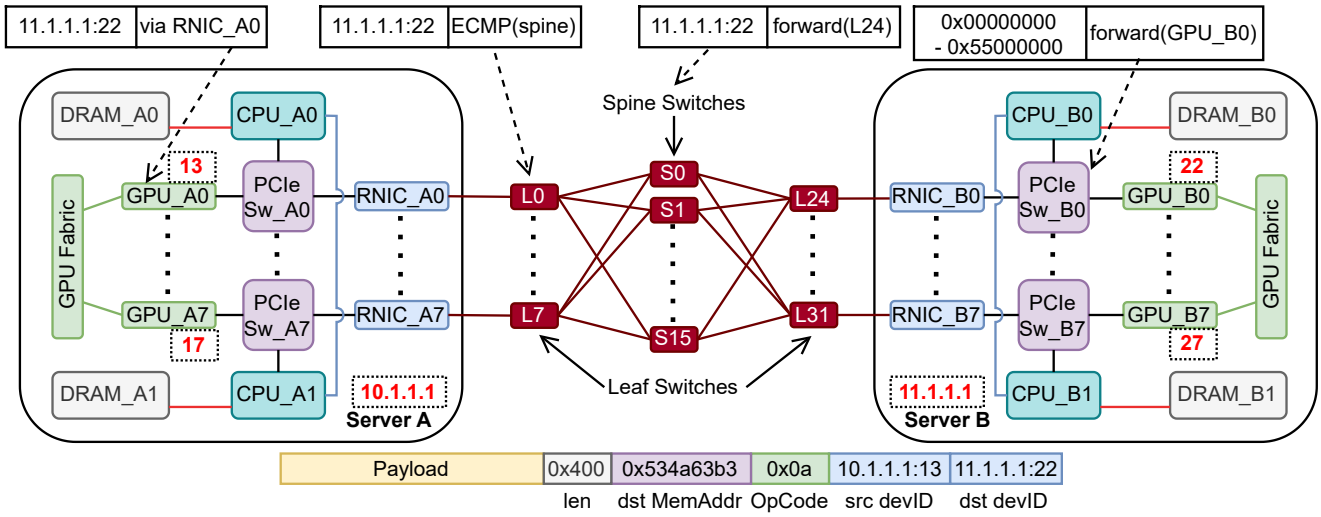

Figure 5: Schematic blueprint for addressing, routing and forwarding of the data transfer from `GPU_A0` to `GPU_B0`. Global server IDs and local device IDs are shown in dotted boxes. Match-action routing entries are at the top.

loss of connectivity to `GPU_B0` when `RNIC_B0` fails, despite of possible alternative paths (e.g. via `RNIC_B7`). This is because GPUs (and other devices) map memory to their nearest RNIC, which serves as the gateway for remote access via RDMA QPs. The Ethernet network routes RoCEv2 packets using the destination RNIC's IP, effectively tying each device's accessibility to a specific RNIC. Thus, accessing a given device requires targeting the RNIC where its memory is registered.

3 Our Vision: Extending the Network Layer Beyond NICs

To address the problems identified in §2 and to better support the emerging device-to-device communication, we argue that we need to re-architect the (AI) datacenter network to treat the intra-host devices as the ultimate network endpoints. To this end, we envision a paradigm where the intra-host and inter-host networks are no longer independent entities but operate together as a *single, unified network*.

In this section, we outline a blueprint for realizing this vision, comprising: (i) a global addressing scheme to uniquely identify intra-host devices across the datacenter; (ii) a routing scheme with routing tables for both intra- and inter-host networks; and (iii) a forwarding mechanism bridging PCIe and GPU fabric interfaces.

Rather than proposing a clean-slate design, our approach emphasizes compatibility and reuse of existing standards wherever possible, and we focus on design concepts rather than implementation specifics. Figure 5 shows the overall blueprint which we explain below considering the example of `GPU_A0` in server A sending data to `GPU_B0` in server B.

3.1 Addressing

Our addressing scheme assigns each device in the datacenter a unique device ID (`DevID`), comprised of two parts: a globally unique server ID (`serverID`) and a local device ID (`localDevID`). For example, in Figure 5, server B has a `serverID` of `11.1.1.1`, and `GPU_B0` has a `localDevID` of `22`, resulting in an `DevID`[4] of `11.1.1.1:22`. The bottom of Figure 5 depicts the schematic[5] of an inter-host Ethernet packet. As noted in §2, data transfers correspond to memory reads/writes at specific addresses. Thus, in addition to the source and destination `DevID`, each packet includes an `OpCode` (indicating a read or write), the destination memory address, and the data length (similar to RoCEv2 "Write Only" packet [12]). Our blueprint does not mandate how the `DevID` is mapped onto Ethernet/IP headers; the only requirement is that inter-host Ethernet switches can parse and match on the destination `DevID`. In summary, this addressing scheme provides unique identification of intra-host device endpoints such as `GPU_A0` and `GPU_B0` across the datacenter, decoupling endpoint addressing from the packet's routing path.

3.2 Routing

To facilitate routing, a local routing agent on each server computes intra-host routes and costs from each gateway RNIC to every device within the server. This information is shared with a logically centralized routing controller, which computes the complete end-to-end path and installs routing rules on GPUs, PCIe switches, RNICs, and Ethernet switches.

[4]Conceptually similar to per-device IP address with a common mask.
[5]Our simplified packet structure doesn't consider transport-level semantics.

For example, the computed route from GPU_A0 to GPU_B0 may traverse: GPU_A0→PCIeSw_A0→RNIC_A0→L0→[S0..S15]→L24→RNIC_B0→PCIeSw_B0→GPU_B0. On GPU_A0, the gateway for destination 11.1.1.1:22 is set as RNIC_A0. Leaf switch L0 uses ECMP to forward packets to the spine switches, which in turn forwards to leaf switch L24, and then to RNIC_B0. Since RNIC_B0 has only one outgoing path, it requires no routing rules. Within the destination host, intra-host routing rules are based on the physical address space (§2.2), and PCIeSw_B0's routing table forwards packets based on the destination memory address to GPU_B0. Overall, this routing approach treats the intra-host and inter-host networks as a single, unified network.

3.3 Forwarding

In our example, since RNIC_A0 is set as the gateway for destination 11.1.1.1:22, it first reads data directly from GPU_A0 via PCIeSw_A0 using PCIe Memory Read TLPs. RNIC_A0 then packetizes the data (see Figure 5) and forwards the packets to leaf switch L0. Ethernet switches forward packets by matching on the destination DevID, rather than the destination IP address, and can support forwarding schemes such as single-path, ECMP, or packet spraying (the example in Figure 5 uses ECMP). Upon arrival at the destination, RNIC_B0 strips the outer header and generates PCIe Memory Write TLPs for GPU_B0, with PCIeSw_B0 forwarding these TLPs based on the memory address.

Forwarding between PCIe and GPU fabric interfaces. We propose a dedicated forwarding hardware block that bridges PCIe and GPU fabric interfaces. The PCIe interface maintains a routing table to identify if a packet's destination memory address belongs to the local GPU and executes the appropriate read/write. Routing rules can forward incoming PCIe packets to the GPU fabric, via the forwarding hardware block, encapsulating them with a GPU fabric header (e.g., NVLink [62]). Similarly, the GPU fabric interface holds a routing table for forwarding packets to the PCIe interface after decapsulation. This design eliminates extra data copies during GPU transit and shifts forwarding from application layer to the network dataplane. We illustrate further in §3.4.

3.4 Case Studies

We now validate our proposed blueprint by applying it to the scenarios from §2.2 and §2.3.

Scenario 1. When GPU0 offloads KV cache to DRAM0 but encounters PCIe bandwidth degradation or path contention (①a, ①b in Figure 2), the local routing agent detects the issue using telemetry from PCIeSw0. It then installs two routing rules (see Figure 6): (i) on GPU0's PCIe interface, to encapsulate transfers to DRAM0 and redirect them to GPU1 over the GPU fabric, and (ii) on GPU1's GPU fabric interface, to decapsulate

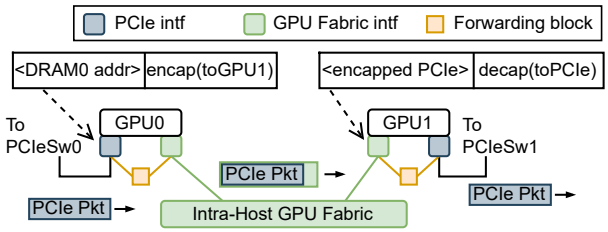

Figure 6: PCIe forwarding for Scenario 1.

and forward these transfers out via its PCIe interface. As a result, Memory-Write TLPs from GPU0 *stream* through GPU1, exit to PCIeSw1, and are ultimately forwarded upstream for writing to DRAM0. The PCIe to GPU fabric forwarding mechanism (§3.3) essentially tunnels PCIe packets through the GPU fabric while being completely transparent to the applications and incurring no memory or control I/O overhead.

Scenario 2. When a GPU's KV cache offload to SSDs is constrained by PCIe bandwidth, it can supplement throughput by utilizing its GPU fabric interface. As an example, consider GPU0 accessing the SSDs attached to PCIeSw7 via the GPU fabric interface (② in Figure 2). To enable this path, the local routing agent installs: (i) a rule on GPU0's PCIe interface to encapsulate traffic for PCIeSw7's SSDs and forward it to GPU7 via the GPU fabric; and (ii) a rule on GPU7's GPU fabric interface to decapsulate and deliver the packets to the SSDs over its PCIe interface. For return traffic, PCIeSw7 is configured to forward packets destined for GPU0 to GPU7, where they are encapsulated on GPU7's PCIe interface and sent to GPU0 over the GPU fabric. Upon arrival, GPU0's GPU fabric interface decapsulates and forwards the packets to its PCIe interface, which performs PCIe read/write operations. By installing similar routing rules, GPU0 can access the remaining 12 SSDs in the system via the GPU fabric. Thus, with only routing rule updates, GPU0 can offload KV cache to all 16 SSDs, using combined PCIe and GPU fabric bandwidth.

Scenario 3. When RNIC_B0 fails during distributed LLM training (Figure 5), the local routing agent on server B detects the failure and updates the intra-host route to 11.1.1.1:22 to use one of the alternative RNICs, say RNIC_B7 via the path: RNIC_B7→PCIeSw_B7→GPU_B7→GPU_B0. This update is communicated to the central routing controller, which triggers end-to-end routing changes. Spine switches now forward packets for 11.1.1.1:22 to leaf switch L31, which then forwards to RNIC_B7 and PCIeSw_B7. PCIeSw_B7 routes packets for GPU_B0's memory to GPU_B7, where a routing rule on GPU_B7's PCIe interface encapsulates and forwards them to GPU_B0 via the GPU fabric. GPU_B0 then decapsulates and handles the PCIe Memory-Write. Similarly, routing rules enable GPU_B0 to send outgoing data via RNIC_B7 through GPU_B7. Thus, even after RNIC_B0 fails, GPU_B0 remains accessible, allowing training to continue uninterrupted.

4 Discussion and Future Work

Implementation. Ethernet switches can parse and match on `DevID` using existing network headers, for example, embedding src/dst `serverID`s in IPv4 address fields and `localDevID`s as IPv4 options. The forthcoming UltraEthernet specification [7] offers an opportunity to standardize host device addressing. Routing and forwarding by physical memory address are already supported by PCIe switches and CPU I/O root complexes. This functionality just needs to become runtime-programmable. The PCIe to GPU fabric forwarding block (§3.3) will require additional hardware support; the upcoming UALink accelerator fabric specification [22] could help standardize this capability.

Alternative Routing Designs. Our blueprint's centralized controller is effective for illustration, but a distributed architecture offers a more scalable alternative. This design would modify how the local routing agent disseminates the intra-host routes and their costs. Instead of reporting to a centralized controller, the agent would advertise them directly to the network fabric using a BGP-like protocol. This would inform more sophisticated, multi-path forwarding decisions directly within the network, enabling dynamic load balancing and bandwidth aggregation.

End-to-end semantics. Similar to the early Internet [5, 43], our blueprint provides a foundational datagram service for addressing, routing, and forwarding packets to host device endpoints. This minimalist packet delivery model serves as a substrate on which more sophisticated end-to-end semantics can be effectively layered. Key directions for future work include: (i) designing new or adapting existing network APIs (such as POSIX sockets) to enable applications to seamlessly address and interact with intra-host device endpoints across the datacenter; (ii) establishing connection-oriented semantics for application multiplexing, end-to-end reliability, flow and congestion control.

Security and Isolation. Our vision of treating intra-host devices as first-class network endpoints indeed introduces important security and isolation considerations, particularly in shared, multi-tenant environments. The RNICs, which serve as gateways between the intra- and inter-host networks, are the natural place to enforce policy. Building on the end-to-end semantics discussed previously, foundational security primitives like firewall rules and access control could be implemented on the RNICs to prevent unauthorized device access. However, ensuring robust performance isolation is more complex due to the differing scheduling schemes across the intra-host and inter-host networks. Devising mechanisms for end-to-end performance guarantees in this unified model is a significant challenge, and we hope this proposal encourages the research community to work towards this important goal.

5 Related Work

DUA [53] proposes a global addressing scheme and interface for FPGA applications to access datacenter resources. While our addressing is inspired by DUA, our blueprint extends addressing into the inter-host network, providing true device endpoint awareness and enabling efficient handling of network failures and load balancing.

NCCL's PXN [38] allows a GPU to communicate via a different RNIC by transiting through another GPU, while FuseLink [49] enables a GPU to aggregate bandwidth across all RNICs (via transit GPUs) during LLM training. Both are application-layer solutions addressing issues such as cross-rail traffic and limited inter-server bandwidth, but suffer from the limitations described in §2.2 and do not fundamentally elevate host devices as network endpoints. By unifying intra- and inter-host networks, our blueprint offers a principled framework for addressing these challenges at the network layer.

Significant work also exists on endpoint network (I/O) stacks for host devices [24, 53, 54], including GPU-driven I/O stacks [15, 19, 46], which are complementary to our approach and help provide end-to-end semantics. NTSocks [18] enables applications to communicate via DMA directly over rack-scale PCIe networks [17], and is orthogonal to our work.

In summary, our blueprint is fundamentally different from prior approaches: it unifies intra-host and inter-host networks to make intra-host devices first-class network endpoints, serving as a framework for clean, principled network-layer solutions rather than point solutions that work around current hardware limitations.

6 Conclusion

In this paper, we advocate rethinking datacenter network architecture to treat host devices as first-class network endpoints thereby expanding the datacenter network layer to encompass the intra-host network. We present the first steps through a blueprint that treats intra- and inter-host networks as a single unified network and lays the groundwork for a rich research agenda. We hope that this timely proposal will help shape emerging standards like UALink and UltraEthernet towards realizing this unified vision.

Acknowledgements

We thank the anonymous reviewers and our shepherd Costin Raiciu for their valuable feedback and guidance. This work was supported in part by ACE, one of the seven centers in JUMP 2.0, a Semiconductor Research Corporation (SRC) program sponsored by DARPA. It was also supported in part by NSF CNS-2211383.

References

[1] Saksham Agarwal, Rachit Agarwal, Behnam Montazeri, Masoud Moshref, Khaled Elmeleegy, Luigi Rizzo, Marc Asher de Kruijf, Gautam Kumar, Sylvia Ratnasamy, David Culler, and Amin Vahdat. 2022. Understanding Host Interconnect Congestion. In *Proceedings of HotNets*.

[2] Saksham Agarwal, Arvind Krishnamurthy, and Rachit Agarwal. 2023. Host Congestion Control. In *Proceedings of SIGCOMM*.

[3] Alex Aizman, Gavin Maltby, and Thomas Breuel. 2019. High Performance I/O For Large Scale Deep Learning. In *Proceedings of IEEE Big Data*.

[4] Matt Bowman and Jeremy Baumgartner. 2023. Grand Teton Systems Overview. https://www.youtube.com/watch?v=fmXfWad-NiA.

[5] V. Cerf and R. Kahn. 1974. A Protocol for Packet Network Intercommunication. *IEEE Transactions on Communications* 22, 5 (1974).

[6] Google Cloud. 2025. Networking and GPU Machines | Compute Engine Documentation. https://cloud.google.com/compute/docs/gpus/gpu-network-bandwidth.

[7] Ultra Ethernet Consortium. 2025. Ultra Ethernet Consortium (UEC) Launches Specification 1.0 Transforming Ethernet for AI and HPC at Scale.

[8] DeepSeek-AI. 2025. DeepSeek-V3 Technical Report. arXiv:2412.19437 [cs]

[9] Dell. 2023. A State-of-the-Art Data Center for Large-Scale AI. https://www.dell.com/en-us/blog/a-state-of-the-art-data-center-for-large-scale-ai/.

[10] Yangtao Deng, Xiang Shi, Zhuo Jiang, Xingjian Zhang, Lei Zhang, Zhang Zhang, Bo Li, Zuquan Song, Hang Zhu, Gaohong Liu, Fuliang Li, Shuguang Wang, Haibin Lin, Jianxi Ye, and Minlan Yu. 2025. Minder: Faulty Machine Detection for Large-scale Distributed Model Training. In *Proceedings of NSDI*.

[11] Jianbo Dong, Kun Qian, Pengcheng Zhang, Zhilong Zheng, Liang Chen, Fei Feng, Yichi Xu, Yikai Zhu, Gang Lu, Xue Li, Zhihui Ren, Zhicheng Wang, Bin Luo, Peng Zhang, Yang Liu, Yanqing Chen, Yu Guan, Weicheng Wang, Chaojie Yang, Yang Zhang, Man Yuan, Hanyu Zhao, Yong Li, Zihan Zhao, Shan Li, Xianlong Zeng, Zhiping Yao, Binzhang Fu, Ennan Zhai, Wei Lin, Chao Wang, and Dennis Cai. 2025. Evolution of Aegis: Fault Diagnosis for {AI} Model Training Service in Production. In *Proceedings of NSDI*.

[12] NVIDIA Developer Forum. 2021. Why Do Lossy RoCE Accelerations Change RDMA Write {First, Middle, Last} to RDMA Write Only? https://forums.developer.nvidia.com/t/why-do-lossy-roce-accelerations-change-rdma-write-first-middle-last-to-rdma-write-only/205952.

[13] Adithya Gangidi, Rui Miao, Shengbao Zheng, Sai Jayesh Bondu, Guilherme Goes, Hany Morsy, Rohit Puri, Mohammad Riftadi, Ashmitha Jeevaraj Shetty, Jingyi Yang, Shuqiang Zhang, Mikel Jimenez Fernandez, Shashidhar Gandham, and Hongyi Zeng. 2024. RDMA over Ethernet for Distributed Training at Meta Scale. In *Proceedings of SIGCOMM*.

[14] GigaByte. 2023. G893-ZD1-AAX3 GPU Server. https://www.gigabyte.com/Enterprise/GPU-Server/G893-ZD1-AAX3.

[15] Massimo Girondi, Mariano Scazzariello, Gerald Q. Maguire, and Dejan Kostić. 2024. Toward GPU-centric Networking on Commodity Hardware. In *Proceedings of EdgeSys*.

[16] Christian Hopps. 2000. Analysis of an Equal-Cost Multi-Path Algorithm. RFC 2992.

[17] Wentao Hou, Jie Zhang, Zeke Wang, and Ming Liu. 2024. Understanding Routable PCIe Performance for Composable Infrastructures. In *Proceedings of NSDI*.

[18] Yibo Huang, View Profile, Yukai Huang, View Profile, Ming Yan, View Profile, Jiayu Hu, View Profile, Cunming Liang, View Profile, Yang Xu, View Profile, Wenxiong Zou, View Profile, Yiming Zhang, View Profile, Rui Zhang, View Profile, Chunpu Huang, View Profile, Jie Wu, and View Profile. 2022. An Ultra-Low Latency and Compatible PCIe Interconnect for Rack-Scale Communication. In *Proceedings of CoNEXT*.

[19] Changho Hwang, KyoungSoo Park, Ran Shu, Xinyuan Qu, Peng Cheng, and Yongqiang Xiong. 2023. ARK: GPU-driven Code Execution for Distributed Deep Learning. In *Proceedings of NSDI*.

[20] Sylvain Jeaugey. 2022. Comment on 'Abnormal recovery problem when GPU node fails', Issue #989. GitHub. https://github.com/NVIDIA/nccl/issues/989#issuecomment-1361225583 Accessed: 2025-07-02.

[21] Ziheng Jiang, Haibin Lin, Yinmin Zhong, Qi Huang, Yangrui Chen, Zhi Zhang, Yanghua Peng, Xiang Li, Cong Xie, Shibiao Nong, Yulu Jia, Sun He, Hongmin Chen, Zhihao Bai, Qi Hou, Shipeng Yan, Ding Zhou, Yiyao Sheng, Zhuo Jiang, Haohan Xu, Haoran Wei, Zhang Zhang, Pengfei Nie, Leqi Zou, Sida Zhao, Liang Xiang, Zherui Liu, Zhe Li, Xiaoying Jia, Jianxi Ye, Xin Jin, and Xin Liu. 2024. MegaScale: Scaling Large Language Model Training to More Than 10,000 GPUs. In *Proceedings of NSDI*.

[22] Nathan Kalyanasundharam. 2025. UALink 200G 1.0 Specification. https://ualinkconsortium.org/.

[23] Chetan Kapoor, Alexandru Costin, and Belinda Zeng. 2023. A Deep Dive on AWS Infrastructure Powering the Generative AI Boom. https://d1.awsstatic.com/events/Summits/reinvent2023/CMP201_A-deep-dive-on-AWS-infrastructure-powering-the-generative-AI-boom.pdf.

[24] Sangman Kim, Seonggu Huh, Xinya Zhang, Yige Hu, Amir Wated, Emmett Witchel, and Mark Silberstein. 2014. GPUnet: Networking Abstractions for GPU Programs. In *Proceedings of OSDI*.

[25] Woosuk Kwon, Zhuohan Li, Siyuan Zhuang, Ying Sheng, Lianmin Zheng, Cody Hao Yu, Joseph Gonzalez, Hao Zhang, and Ion Stoica. 2023. Efficient Memory Management for Large Language Model Serving with PagedAttention. In *Proceedings of SOSP*.

[26] Woosuk Kwon, Zhuohan Li, Siyuan Zhuang, Ying Sheng, Lianmin Zheng, Cody Hao Yu, Joseph Gonzalez, Hao Zhang, and Ion Stoica. 2023. Efficient Memory Management for Large Language Model Serving with PagedAttention. In *Proceedings of SOSP*.

[27] ChonLam Lao, Minlan Yu, Aditya Akella, Jiamin Cao, Yu Guan, Pengcheng Zhang, Zhilong Zheng, Yichi Xu, Ennan Zhai, Dennis Cai, and Jiaqi Gao. 2025. TrainMover: An Interruption-Resilient and Reliable ML Training Runtime. arXiv:2412.12636 [cs.DC]

[28] Wonbeom Lee, Jungi Lee, Junghwan Seo, and Jaewoong Sim. 2024. InfiniGen: Efficient Generative Inference of Large Language Models with Dynamic KV Cache Management. In *Proceedings of OSDI*.

[29] Lenovo. 2023. ThinkSystem SR685a V3. https://pubs.lenovo.com/sr685a-v3/.

[30] Patrick Lewis, Ethan Perez, Aleksandra Piktus, Fabio Petroni, Vladimir Karpukhin, Naman Goyal, Heinrich Küttler, Mike Lewis, Wen-tau Yih, Tim Rocktäschel, Sebastian Riedel, and Douwe Kiela. 2021. Retrieval-Augmented Generation for Knowledge-Intensive NLP Tasks. arXiv:2005.11401 [cs]

[31] Kefei Liu, Zhuo Jiang, Jiao Zhang, Haoran Wei, Xiaolong Zhong, Lizhuang Tan, Tian Pan, and Tao Huang. 2023. Hostping: Diagnosing Intra-host Network Bottlenecks in RDMA Servers. In *Proceedings of NSDI*.

[32] Vincent Liu, Daniel Halperin, Arvind Krishnamurthy, and Thomas Anderson. 2013. F10: A Fault-Tolerant Engineered Network. In *Proceedings of NSDI*.

[33] MinIO. 2025. *NVIDIA GPUDirect Storage and MinIO AIStor: Unlocking Efficiency for GPU-Powered AI Workloads.* https://blog.min.io/nvidia-

gpudirect-storage-and-aistor/
[34] Deepak Narayanan, Mohammad Shoeybi, Jared Casper, Patrick LeGresley, Mostofa Patwary, Vijay Korthikanti, Dmitri Vainbrand, Prethvi Kashinkunti, Julie Bernauer, Bryan Catanzaro, Amar Phanishayee, and Matei Zaharia. 2021. Efficient Large-Scale Language Model Training on GPU Clusters Using Megatron-LM. In *Proceedings of SC*.
[35] NVIDIA. 2020. DGX A100 System Topology. https://docs.nvidia.com/dgx/dgxa100-user-guide/introduction-to-dgxa100.html#dgx-a100-system-topology.
[36] NVIDIA. 2020. GPUDirect Storage. https://docs.nvidia.com/gpudirect-storage/index.html.
[37] NVIDIA. 2021. Magnum IO GPUDirect Storage Overview Guide v1.0.0. https://docs.nvidia.com/cuda/archive/11.4.0/gds/pdf/overview-guide.pdf.
[38] NVIDIA. 2022. Doubling All2all Performance with NVIDIA Collective Communication Library 2.12. https://developer.nvidia.com/blog/doubling-all2all-performance-with-nvidia-collective-communication-library-2-12/.
[39] NVIDIA. 2024. *Storage Architecture*. https://docs.nvidia.com/dgx-superpod/reference-architecture/scalable-infrastructure-h200/latest/storage-architecture.html
[40] NVIDIA. 2025. GPUDirect RDMA. https://docs.nvidia.com/cuda/gpudirect-rdma/index.html.
[41] Pratyush Patel, Esha Choukse, Chaojie Zhang, Aashaka Shah, Íñigo Goiri, Saeed Maleki, and Ricardo Bianchini. 2024. Splitwise: Efficient Generative LLM Inference Using Phase Splitting. In *Proceedings of ISCA*.
[42] PCI-SIG 2022. *PCI Express Base Specification, Revision 6.0*. PCI-SIG.
[43] Jon Postel. 1981. *Internet Protocol*. Request for Comments RFC 791. Internet Engineering Task Force.
[44] Lenovo Press. 2024. ThinkSystem NVIDIA H100 PCIe Gen5 GPUs Product Guide. https://lenovopress.lenovo.com/lp1732-thinksystem-nvidia-h100-pcie-gen5-gpu.
[45] Ruoyu Qin, Zheming Li, Weiran He, Jialei Cui, Feng Ren, Mingxing Zhang, Yongwei Wu, Weimin Zheng, and Xinran Xu. 2025. Mooncake: Trading More Storage for Less Computation — A KVCache-centric Architecture for Serving LLM Chatbot. In *Proceedings of FAST*.
[46] Zaid Qureshi, Vikram Sharma Mailthody, Isaac Gelado, Seungwon Min, Amna Masood, Jeongmin Park, Jinjun Xiong, C. J. Newburn, Dmitri Vainbrand, I-Hsin Chung, Michael Garland, William Dally, and Wen-mei Hwu. 2023. GPU-Initiated On-Demand High-Throughput Storage Access in the BaM System Architecture. In *Proceedings of ASPLOS*.
[47] Samyam Rajbhandari, Olatunji Ruwase, Jeff Rasley, Shaden Smith, and Yuxiong He. 2021. ZeRO-infinity: Breaking the GPU Memory Wall for Extreme Scale Deep Learning. In *Proceedings of SC*.
[48] Zebin Ren, Krijn Doekemeijer, Tiziano De Matteis, Christian Pinto, Radu Stoica, and Animesh Trivedi. 2025. An I/O Characterizing Study of Offloading LLM Models and KV Caches to NVMe SSD. In *Proceedings of CHEOPS*.
[49] Zhenghang Ren and Yuxuan Li. 2025. Enabling Efficient GPU Communication over Multiple NICs with FuseLink. In *Proceedings of OSDI*.
[50] Amazon Web Services. 2020. Amazon EC2 P4d Instances Deep Dive | AWS Compute Blog. https://aws.amazon.com/blogs/compute/amazon-ec2-p4d-instances-deep-dive/.
[51] Noam Shazeer, Azalia Mirhoseini, Krzysztof Maziarz, Andy Davis, Quoc Le, Geoffrey Hinton, and Jeff Dean. 2017. Outrageously Large Neural Networks: The Sparsely-Gated Mixture-of-Experts Layer. arXiv:1701.06538 [cs]
[52] Ying Sheng, Lianmin Zheng, Binhang Yuan, Zhuohan Li, Max Ryabinin, Beidi Chen, Percy Liang, Christopher Ré, Ion Stoica, and Ce Zhang. 2023. FlexGen: High-Throughput Generative Inference of Large Language Models with a Single GPU. In *Proceedings of ICML*.
[53] Ran Shu, Peng Cheng, Guo Chen, Zhiyuan Guo, Lei Qu, Yongqiang Xiong, Derek Chiou, and Thomas Moscibroda. 2019. Direct Universal Access: Making Data Center Resources Available to FPGA. In *Proceedings of NSDI*.
[54] Athinagoras Skiadopoulos, Zhiqiang Xie, Mark Zhao, Qizhe Cai, Saksham Agarwal, Jacob Adelmann, David Ahern, Carlo Contavalli, Michael Goldflam, Vitaly Mayatskikh, Raghu Raja, Daniel Walton, Rachit Agarwal, Shrijeet Mukherjee, and Christos Kozyrakis. 2024. High-Throughput and Flexible Host Networking for Accelerated Computing. In *Proceedings of OSDI*.
[55] Supermicro. 2023. AS-8125GS-TNHR. https://www.supermicro.com/manuals/superserver/8U/MNL-2598.pdf.
[56] The Llama Team. 2024. The Llama 3 Herd of Models. arXiv:2407.21783 [cs]
[57] Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N. Gomez, Lukasz Kaiser, and Illia Polosukhin. 2023. Attention Is All You Need. arXiv:1706.03762 [cs]
[58] Abhishek Vijaya Kumar, Gianni Antichi, and Rachee Singh. 2025. Aqua: Network-Accelerated Memory Offloading for LLMs in Scale-Up GPU Domains. In *Proceedings of ASPLOS*.
[59] vLLM. 2025. Distributed Inference and Serving. https://docs.vllm.ai/en/latest/serving/distributed_serving.html.
[60] Midhul Vuppalapati, Saksham Agarwal, Henry Schuch, Baris Kasikci, Arvind Krishnamurthy, and Rachit Agarwal. 2024. Understanding the Host Network. In *Proceedings of SIGCOMM*.
[61] Zhuang Wang, Zhen Jia, Shuai Zheng, Zhen Zhang, Xinwei Fu, T. S. Eugene Ng, and Yida Wang. 2023. GEMINI: Fast Failure Recovery in Distributed Training with In-Memory Checkpoints. In *Proceedings of SOSP*.
[62] WikiChip. 2022. NVLink - Nvidia. https://en.wikichip.org/wiki/nvidia/nvlink.
[63] Minhui Xie, Youyou Lu, Yangyang Feng, and Jiwu Shu. 2024. A Recommendation Model Inference System Based on GPU Direct Storage Access Architecture. *Journal of Computer Research and Development* 61, 3 (2024), 589–599. doi:10.7544/issn1000-1239.202330402
[64] Takeshi Yoshimura, Tatsuhiro Chiba, Manish Sethi, Daniel Waddington, and Swaminathan Sundararaman. 2025. Speeding up Model Loading with fastsafetensors. arXiv:2505.23072v1 [cs.DC]
[65] Yinmin Zhong, Shengyu Liu, Junda Chen, Jianbo Hu, Yibo Zhu, Xuanzhe Liu, Xin Jin, and Hao Zhang. 2024. DistServe: Disaggregating Prefill and Decoding for Goodput-optimized Large Language Model Serving. In *Proceedings of OSDI*.