# Unlocking Software-defined GPU Fabric Scheduling in the LLM Era

Danyang Chen[1*] Yufeng Gu[2*] Yibo Huang[2†] Chengxuan Pei[1]
Peichun Hua[1] Yang Zhou[3] Yunming Xiao[1]

[1] The Chinese University of Hong Kong, Shenzhen [2] University of Michigan [3] University of California, Davis

## Abstract

Large language model (LLM) systems increasingly rely on techniques such as prefill–decode disaggregation, KV-cache offloading, and computation–communication overlap. These optimizations often treat GPU interconnects as best-effort substrates, overlooking contention across shared PCIe, NVLink, and RDMA fabrics. We characterize GPU-fabric contention in LLM workloads and find that hardware arbitration can impose implicit priorities across traffic classes, causing communication to interfere with computation and other data movements. In particular, GPU-to-GPU NVLink communication can interfere with GPU memory access, while GPU-direct RDMA can contend with host–device transfers over PCIe. Based on these observations, we advocate a vision for software-defined GPU fabric scheduling and present GPUWeaver, a context-aware communication scheduler that monitors application progress and fabric state, identifies contention, and dynamically regulates communication injection according to application performance goals. Our study highlights the opportunity to make GPU fabrics actively managed resources for efficient and predictable LLM systems.

## CCS Concepts

• **Computer systems organization** → **Heterogeneous (hybrid) systems**; **Interconnection architectures**; • **Networks** → *Network performance analysis*; **Data center networks**.

## Keywords

GPU fabrics, datacenter systems, heterogeneous accelerators, machine learning systems, communication scheduling, resource contention, PCIe, NVLink, RDMA, LLM serving

*Danyang Chen and Yufeng Gu contributed equally to this research.
†Yibo Huang is the corresponding author.



*APSys '26, Bangkok, Thailand*



ACM ISBN 979-8-4007-2912-6/26/09
https://doi.org/10.1145/3838177.3841731

**ACM Reference Format:**
Danyang Chen, Yufeng Gu, Yibo Huang, Chengxuan Pei, Peichun Hua, Yang Zhou, and Yunming Xiao. 2026. Unlocking Software-defined GPU Fabric Scheduling in the LLM Era. In *ACM SIGOPS Asia-Pacific Workshop on Systems (APSys '26), September 17–18, 2026, Bangkok, Thailand.* ACM, New York, NY, USA, 8 pages. https://doi.org/10.1145/3838177.3841731

## 1 Introduction

LLMs are pushing GPU clusters into a new regime where the GPU fabric is a first-class system resource [32, 36]. Unlike traditional DL workloads with moderate transfers, LLM pipelines generate high-volume traffic across heterogeneous links (NVLink, PCIe, and GPUDirect RDMA) [31, 44]. LLMs also expand the working set from "small and mostly static" to "large and stateful" (Table 1): beyond GB-scale parameters, they maintain hundreds of GBs of runtime state (e.g., KV cache) that must be created and accessed at runtime [20, 34]. This breaks the traditional assumption that a single GPU (e.g., H100) can easily hold a whole AI model [8, 30, 36]. As a result, LLM workloads increasingly depend on how efficiently the system places and transfers parameters and state across GPUs and hosts—requiring a fabric that offers not just bandwidth, but also predictability and hardware efficiency.

This shift has driven LLM infrastructure techniques such as disaggregation [33, 44], parallelism [28, 36], compute-communication overlap [43], and collective communication optimization [16, 45]. However, these systems often treat the GPU fabric as a best-effort black box. This creates a context gap between what LLM workloads need—service level objectives (SLO) and phase-specific goals in LLM pipelines—and what the fabric actually provides: shared hardware resources with complex contention behaviors (Figure 1). Without fabric-aware coordination, LLM runtimes could show high nominal GPU activity yet still experience utilization cliffs, unpredictable slowdown, and costly overprovisioning in AI datacenters.

In this paper, we argue for a vision we call software-defined GPU fabric scheduling: modeling GPU fabric use as an explicit scheduling problem—not an emergent side effect—so LLM jobs and their pipeline phases can coordinate to optimize global objectives. Basically, LLM tasks (or phases) are not uniformly urgent: tasks with slack (e.g., background

| Model / HW | Params | Runtime State | Takeaway |
|---|---|---|---|
| ResNet-50(DL) | 45MB | Act@2048: 4GB | Small & static |
| Llama3.1-405B(LLM) | 810GB | KV@128K: 123GB | Large & stateful |
| Qwen3-Max(LLM) | >2000GB | KV@262K: 88GB | Large & stateful |
| H100(GPU) | L2 50MB | HBM 80GB | – |

**Table 1: DL vs. LLM footprint shift (BF16/FP16).**

state movement) can yield GPU fabric resources to latency-critical tasks with tight deadlines, enabling SLO-aware sharing rather than best-effort contention. Realizing this vision requires elevating the GPU fabric from "plumbing" to a managed resource, much like how OS schedulers manage CPU and memory rather than leaving it to uncoordinated processes. Concretely, GPU fabric should expose control knobs (e.g., NVLink pacing) and valuable signals (e.g., GPU memory bandwidth usage) to a scheduler, and the scheduler should be able to automatically adjust control knobs reacting to relevant signals, driven by the objectives of LLM tasks. This would allow LLM tasks to share GPU fabric resources cooperatively, delivering predictable LLM performance without sacrificing high GPU utilization. Prior work has exposed otherwise rigid network mechanisms for software-defined traffic control [24] and scheduling of RNIC resources [14]; GPUWeaver extends this idea to heterogeneous GPU fabrics.

Achieving this vision raises several challenges. First, we must explain why current LLM deployments struggle: identify when LLM jobs or phases violate their goals and attribute slowdowns to specific root causes in GPU fabric resource contention across LLM tasks and phases. This requires a deep understanding on the intrinsic microarchitectural behaviors of GPU fabrics under realistic LLM I/O patterns. Second, we must connect root causes to reusable primitives: determine which signals are predictive and practical to gather, and which control knobs are feasible to enforce. Unlike CPU scheduling, not all GPU fabric resources expose direct actionable controls; an effective design must combine direct controls where available with indirect knobs such as pacing and traffic shaping at appropriate software boundaries. Finally, we need a mapping layer that translates global and LLM task specific objectives (e.g., SLOs and phase properties) into timely microarchitecture-level scheduling strategies that can be enforced for existing LLM stacks transparently.

We present GPUWeaver, a lightweight runtime framework toward this vision, grounded in two kinds of GPU fabric contentions that increasingly dominate LLM deployments (§3): high-bandwidth memory (HBM) contention, where NVLink-driven accesses interfere with local GPU memory traffic and silently throttle compute, and PCIe contention, where GPUDirect RDMA competes with GPU⇔CPU data movement (e.g., KV cache CPU offload), breaking overlap assumptions and amplifying tail latency with degraded throughput. Based on these insights, GPUWeaver (1) characterizes

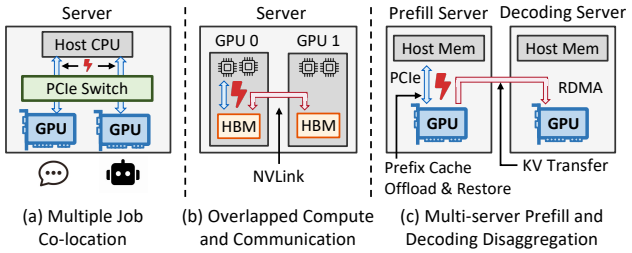


**Figure 1: Resource contentions in LLM deployments.**

GPU memory and PCIe contention with controlled measurements, exposing their microarchitectural root causes and implicit priority behaviors (§3); (2) provides a closed-loop runtime that lets applications register prioritized communication flows, observes fabric and application signals, and adjusts software-visible control knobs to regulate communication injection (§4.1); and (3) demonstrates an integrated vLLM+Mooncake case study using Qwen3-8B and request shapes from a real Codex/SWE-bench Pro trace, reducing mean TTFT by 2.4–6.1% and the PD KV transfer flow time by 19.4–34.4% (§4.2).

This work does **not** raise any ethical issues.

## 2 Background

In this section, we further demonstrate the trend of GPU fabric resource sharing in LLM stacks, discuss the major limitations of existing efforts, and explore an operating point—software-defined GPU fabric scheduling.

### 2.1 GPU Fabric Sharing in LLM Workloads

To improve hardware utilization, modern AI and LLM stacks increasingly adopt system-level optimizations, following a broader trend toward disaggregated AI serving [15, 26, 44]. These efforts include offloading the runtime state (e.g., KV cache) from GPU high bandwidth memory (HBM) to host memory [3, 18, 37, 40] or remote storage [46, 47], colocating multiple LLM instances on the same multi-GPU server (e.g., a chatbot alongside a document assistant in Figure 1(a)) [5, 7, 41], overlapping compute and communication for LLM operator parallelisms (Figure 1(b)) [25, 42, 43], and disaggregating inference into separate prefill and decode workers that may run on different GPUs or even different servers (Figure 1(c)) [33, 44]. These efforts inherently lead to *GPU fabric sharing* across tasks with various objectives: KV transfer between two servers are typically *latency-critical* (the lower the latency, the better), while background activities such as KV cache offloading are often best-effort yet still consume substantial GPU fabric resources.

When co-existing tasks share the same GPU fabric, they inevitably interfere with one another as they concurrently contend for shared hardware resources. Specifically, GPU

HBM can be accessed by local GPU compute units for LLM operator compute, or by NVLink for moving state to a neighboring GPU; a PCIe link is shared by GPU↔CPU transfers and GPU↔RDMA NIC (RNIC) transfers via GPUDirect RDMA, which extends RDMA-based direct memory access [10, 11, 23] to GPU memory. When LLM tasks use them concurrently, these components can independently cause resource contention. For example, a latency-sensitive RDMA-based KV transfer flow between prefill and decode servers may be colocated with a periodic best-effort KV cache offload from GPU to CPU memory [34, 38, 40]. The offload traffic can consume substantial PCIe bandwidth, thereby delaying the PD KV transfer flow and inflating serving latency such as TTFT.

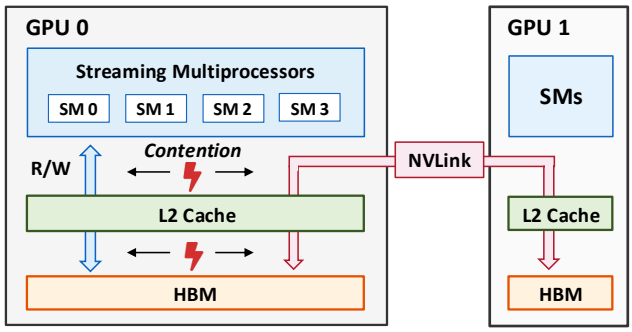


**Figure 2: Contention between memory access and NVLink transfer.**

## 2.2 Why Existing Approaches Fall Short

Despite rapid progress in LLM systems, most systems remain *workload-centric*. They optimize overlap and execution scheduling within a job [2, 9, 21], and thus lack a principled design to orchestrate various LLM tasks. This exposes three limitations:

**(1) Limited fabric-centric control.** Existing schedulers and software-defined networking abstractions primarily reason about *which* compute/communication stages to run and *when* [13, 29, 39], but rarely control *how* different traffic classes are injected into shared arbitration domains (e.g., PCIe switch hierarchies). As a result, they cannot explicitly coordinate competing flows on the shared fabric and must rely on indirect workload-level decisions [1].

**(2) Opaque and largely immutable arbitration.** Once requests are injected into a shared domain (e.g., the PCIe hierarchy or GPU memory subsystem), effective service order and priority are largely determined by hardware mechanisms. Such opaque resource interactions have also been observed in RDMA subsystems [19] and unified-memory edge platforms [27]. Practical Quality-of-Service (QoS) controls are limited and often insufficient to enforce application-level criticality under contention [35, 38].

**(3) Reactive, coarse-grained workarounds.** Prior approaches mainly mitigate interference by blocking overlap, throttling at coarse granularity, or applying stream-level prioritization, rather than *scheduling the fabric* itself to recover latent bandwidth and align resource allocation with demand [1, 4, 21, 22].

**A new operating point: GPU fabric scheduling.** These limitations motivate a *software-defined GPU fabric scheduling framework* called GPUWeaver that (i) allows LLM tasks to express explicit intent (e.g., performance objectives and QoS level), (ii) timely perceives impending resource contentions and reasons about their root causes, and (iii) applies scheduling policies through software-visible control knobs to improve fabric sharing efficiency.

# 3 Fabric Contentions

We present two forms of fabric contention that we observe in modern GPU clusters, along with controlled measurements that expose their microarchitectural behavior and severity.

*Use of AI tools.* OpenAI Codex with was used to assist in writing benchmarking and plotting scripts for the experiments reported in this paper. The authors reviewed and tested all AI-assisted code and independently verified the experimental data, plots, and reported results.

## 3.1 GPU Memory Contention

Computation in the streaming multiprocessor (SM) involves memory requests for GPU's HBM. Inter-GPU communication is enabled by NVLink, where dedicated DMA copy engines are responsible for transferring data between the HBM and the NVLink interface in both directions. Consequently, computation and communication inject traffic into the same on-chip interconnect and off-chip HBM, leading to resource contention [6], as illustrated in Figure 2.

**Root cause**: Contention emerges because memory requests issued by SMs and DMA transfers from copy engines compete for shared resources within a unified arbitration domain. First, the interconnect fabric of L2 cache provides limited bandwidth and constrained arbitration capacity, such that simultaneous bursts from SMs and DMA engines are competing at L2 fabric. Second, memory controllers maintain request queues with fixed capacity. Co-locating SM and DMA transactions within these queues can generate back-pressure and pipeline stalls as occupancy increases, thereby degrading effective throughput for both traffic streams.

**Contention characterization.** We characterize GPU memory contention by concurrently executing a memory access

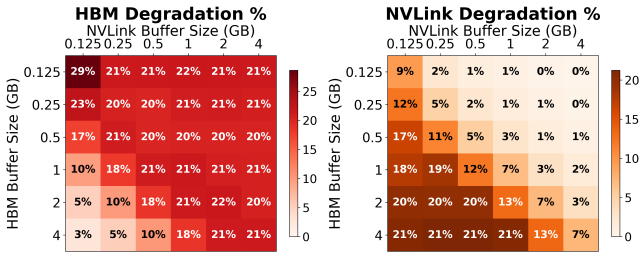


**Figure 3: HBM and NVLink effective bandwidth degradation under contention.**

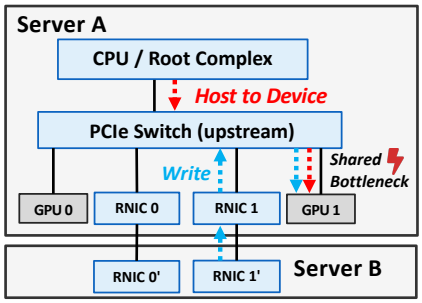


**Figure 4: PCIe contention topology. Blue: NIC→GPU GPUDirect RDMA write. Red: CPU→GPU H2D. Both contend on the same GPU-side PCIe segment.**

kernel and an NVLink communication kernel on a DGX–H100 server. Both kernels are launched in parallel on GPU A using separate CUDA streams to enable overlap. The memory access kernel generates interleaved read and write operations to contiguous memory locations, specifically loading a 128-byte cache line from HBM and subsequently writing it back to the same region, thereby sustaining streaming memory traffic. The NVLink communication kernel invokes the CUDA runtime API `cudaMemcpyPeerAsync` to perform a bidirectional peer-to-peer data transfer between GPU A and GPU B.

We sweep both memory access and NVLink transfer buffer sizes from 128 MB to 4 GB. The bi-directional NVLink transfer uses such buffer size in each direction. Specifically, we evaluate a two-dimensional grid in which the memory access size and the NVLink transfer size independently take values from 128 MB to 4 GB. For each (HBM, NVLink) pair, three scenarios are measured: (1) an isolated HBM baseline with only the memory access kernel active, (2) an isolated NVLink baseline with only the NVLink communication kernel active, and (3) a concurrent execution in which both kernels run simultaneously. Contention is reported as a percentage degradation,

$$\text{Degradation}(\%) = 100 \times \left(1 - \frac{\text{BW}_{\text{contended}}}{\text{BW}_{\text{isolated}}}\right),$$

where normalization is performed against the corresponding isolated baseline for the metric under consideration.

The results are shown in Figure 3. Across the $6 \times 6$ sweep (128 MB–4 GB), the two heatmaps indicate that contention is primarily governed by the temporal overlap between the GPU's local memory access and NVLink transfer.

HBM shows a triangular plateau in the upper-right and near-diagonal regions, where comparable or longer NVLink transfers impose sustained copy-engine pressure and reduce HBM throughput by roughly 20%, while shorter NVLink transfers in the lower-left region cause only about 3–10% loss due to limited overlap. NVLink degradation follows an approximately transposed pattern, reaching around 20% when short transfers are fully overlapped by long-running memory accesses, but falling to 0–1% when the memory kernel finishes early and the transfer proceeds mostly uncontended. Along the matched-duration diagonal for sizes $\geq$ 256 MB, HBM consistently degrades more than NVLink, with NVLink typically limited to about 5–7%, suggesting that NVLink traffic retains a service advantage under full overlap. The anomalous 29% HBM degradation at 128 MB likely reflects small working-set or non–steady-state effects outside the stable streaming regime.

## 3.2 PCIe Contention

Modern GPU servers place GPUs and RDMA NICs under the same PCIe fabric. As Figure 4 shows, two traffic classes may concurrently share the PCIe: (i) *GPU↔CPU* host-device data movement, and (ii) *NIC↔GPU* communication via GPUDirect RDMA (GDR), where the NIC directly reads/writes GPU memory. PCIe provides low bandwidth, and thus is more likely to become a bottleneck when host-device movement and RDMA traffic occur simultaneously. Prior work has also explored PCIe as a low-latency interconnect for rack-scale communication [12], further highlighting the importance of understanding PCIe behavior as systems increasingly rely on it for accelerator and network I/O.

**Root cause: contention on a shared PCIe segment.** As Figure 4 illustrates, the RDMA write stream is initiated by Server B and writes directly into Server A's GPU memory via GDR, while the H2D stream is generated on Server A by its CPU staging data to the GPU. Both streams traverse the same GPU-side PCIe switch/uplink and thus contend on a shared segment. Similarly, RDMA read and D2H follow analogous paths in the reverse directions and also traverse the reverse GPU-side PCIe switch/uplink, leading to contention.

**Contention characterization.** We characterize PCIe contention using a two-server RDMA setup. On Server A, we generate sustained host-device copy traffic using pinned host memory and `cudaMemcpyAsync` to saturate either D2H or

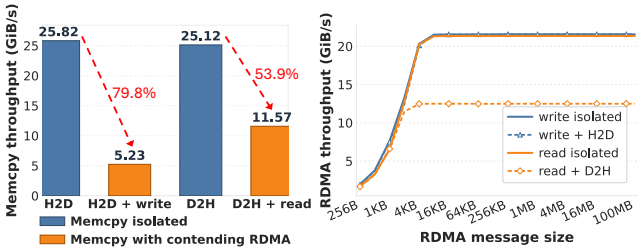


**Figure 5: PCIe contention characterization. Left: large-RDMA impact on host-device copies. Right: RDMA throughput under four cases.**

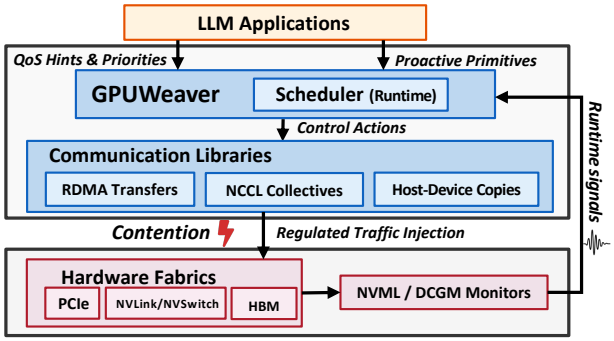


**Figure 6: System overview of GPUWeaver.**

H2D. Concurrently, a remote Server B issues GDR read/write requests to Server A's GPU memory.

We first measure steady-state contention under large RDMA messages. Figure 5 (left) reveals a striking asymmetry: under concurrent GDR write traffic, H2D bandwidth drops from 25.82 GiB/s to 5.23 GiB/s (79.8% degradation); while under GDR read traffic, D2H bandwidth decreases from 25.12 GiB/s to 11.57 GiB/s (53.9% degradation).

Next, we examine the reverse direction by quantifying how sustained host–device transfers affect GDR performance across message sizes. We run four cases: (1) GDR write isolated, (2) GDR write with background H2D, (3) GDR read isolated, and (4) GDR read with background D2H. Figure 5 (right) shows a strong direction-dependent asymmetry: background H2D has negligible impact on GDR write throughput, while background D2H sharply reduces GDR read throughput and stabilizes at ~41.5% degradation when message sizes are ≥8 KB.

Overall, these results reveal an *implicit priority skew on PCIe that varies across traffic types and directions*. This skew originates from hardware design and cannot be changed once the device is fabricated, and may prevent demanding tasks from obtaining sufficient bandwidth. Since the PCIe segments often lie on the critical path of workloads, such skew can amplify end-to-end slowdown though the link is highly utilized. This motivates software mechanisms to flexibly protect demanding tasks under PCIe concurrent traffic.

## 4 GPUWeaver framework

To mitigate the fabric-level contention, we propose GPUWeaver, a lightweight runtime framework. The key idea is to regulate the effective injection behavior of communication flows, controlling how aggressively different traffic classes enter shared fabrics such as PCIe and NVLink.

### 4.1 System Overview

Figure 6 illustrates the GPUWeaver architecture. The runtime sits between user-space LLM applications and the underlying communication libraries, acting as a control layer that observes runtime behavior and regulates communication injection. Applications register communication flows, such as RDMA transfers, collectives, or host-device copies, and attach priorities or performance goals to these flows. GPUWeaver then automatically adjusts how aggressively each flow injects traffic into shared fabrics like PCIe/NVLink.

**Signals:** GPUWeaver employs a closed-loop scheduler that monitors both infrastructure- and application-level signals. Infrastructure signals characterize fabric congestion, including NVML/DCGM PCIe counters, NVLink counters, NIC throughput, queue depth, and GPU memory-bandwidth utilization. Application signals capture execution progress, such as request latency, stage latency, transfer latency, and per-flow backlog. These infrastructure metrics indicate when a link is congested, whereas application-level metrics identify the specific flow adversely affected by the contention.

**Policy:** The scheduler maps these signals and user-provided priorities to per-flow control decisions. If a high-priority flow misses its latency or bandwidth target while the shared fabric is saturated, GPUWeaver shifts the bandwidth budget toward that flow in the next control window. If the high-priority flow is healthy and background backlog grows, GPUWeaver shifts the bandwidth budget back. This policy can be a fixed split, a threshold rule, or a learned controller; GPUWeaver only requires that it output a bandwidth budget or pacing decision for each flow.

**Control knobs:** GPUWeaver implements policy decisions through software-controlled mechanisms. For RDMA traffic, it leverages service levels, NIC QoS, and rate limiting, building on prior observations that RNIC resources can be explicitly exposed and scheduled [14]. For host-device transfers, where CUDA lacks native bandwidth control, GPUWeaver relies on stream priorities, launch gating, chunked copies, and cooperative yield points. Despite backend-specific implementations, all mechanisms share a common objective: regulating the traffic injection rate of each flow into the shared fabric.

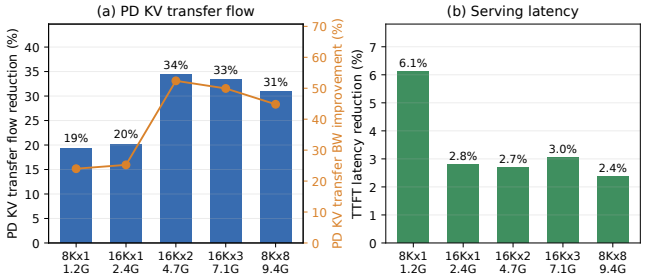


**Figure 7: GPUWeaver improves TTFT and PD KV transfer latency in vLLM + Mooncake.**

## 4.2 Case Study: KV Cache Offloading

To illustrate the feasibility of GPUWeaver, we integrate a preliminary prototype into a real vLLM serving stack with Mooncake, and replay request shapes derived from a real SWE-bench Pro agentic trace collected from Codex workloads [17]. In this setting, a prefill worker sends newly generated KV to the decode worker through GPUDirect RDMA while also offloading prefix KV to CPU. The latency-critical RDMA Write traffic and the background GPU-to-CPU traffic share the same GPU-side PCIe resources.

The serving runtime registers two flows, kv transfer and kv offload. A PD signal opens a protected handoff window, and GPUWeaver chooses how much kv cache offload traffic can run during that window. In our current prototype, the primary control knob is offload delay injection. The policy uses the previous round's PCIe bandwidth, RDMA bandwidth, TTFT, and offload backlog to choose the next round's delay or PCIe bandwidth split. For example, if PD RDMA does not reach its target bandwidth, the next round assigns it a larger PCIe bandwidth budget by delaying or pacing offload more aggressively; if PD transfer is healthy, the next round returns more bandwidth to offload.

Figure 7 evaluates this control loop in the integrated vLLM + Mooncake using Qwen3-8B. The setup emulates PD disaggregation with two logical servers, one GPU per server. We replay several request shapes from the Codex trace and vary the aggregate KV size from 1.2 GB to 9.4 GB. The x-axis label *8K×1*, for example, denotes an 8K-token prompt with one concurrent request. For each shape, we first run the default policy, where kv transfer and offload traffic naturally overlap. We then let GPUWeaver sweep candidate PCIe allocations and select the best policy using the monitored RDMA bandwidth and TTFT from prior runs.

The results show that, across all request shapes, GPUWeaver reduces the latency-critical PD KV transfer flow time by 19.4–34.4%, which in turn reduces mean TTFT by 2.4–6.1%. The smaller TTFT gain is expected because the PD KV transfer flow occupies only a fraction of the TTFT path.

## 5 Conclusion & Future Work

This paper examines fabric-level contention in modern GPU clusters. We identify two common forms of contention: memory read/write versus NVLink communication on HBM access, and GPUDirect RDMA versus host–device transfers on PCIe. Our results show that implicit priority behaviors in these fabrics can cause certain traffic classes to disproportionately throttle others, particularly in emerging LLM workloads that rely on disaggregation, offloading, and compute–communication overlap.

We propose GPUWeaver, a lightweight runtime framework that connects fabric-level signals with application-level goals and regulates communication injection through software-visible control knobs. In an integrated vLLM + Mooncake case study using Qwen3-8B and request shapes from a real Codex/SWE-bench Pro trace, GPUWeaver reduces the PD KV transfer flow time by 19.4–34.4% and mean TTFT by 2.4–6.1%. These results show that fabric contention can be turned from an opaque hardware side effect into a runtime scheduling problem.

Our current prototype still uses a simple scheduling policy and coarse control actions. Future work will refine this policy layer with more detailed per-flow models, faster online adaptation, and richer bandwidth allocation objectives across latency-critical and background traffic. We also plan to extend GPUWeaver beyond the PCIe case study to GPU memory contention, where NVLink communication and local HBM access interact with GPU compute. Together, these directions move toward a software-defined GPU fabric scheduler that can manage PCIe, RDMA, NVLink, and GPU memory pressure under one application-aware control plane.

## Acknowledgements

We would like to thank the anonymous reviewers for their insightful and constructive comments. Yibo Huang is the corresponding author.

## References

[1] Jiamin Cao, Yu Guan, Kun Qian, Jiaqi Gao, Wencong Xiao, Jianbo Dong, Binzhang Fu, Dennis Cai, and Ennan Zhai. 2024. Crux: GPU-Efficient Communication Scheduling for Deep Learning Training. In *Proceedings of the ACM SIGCOMM 2024 Conference*. ACM, Sydney NSW Australia, 1–15. doi:10.1145/3651890.3672239

[2] Chang Chen, Xiuhong Li, Qianchao Zhu, Jiangfei Duan, Peng Sun, Xingcheng Zhang, and Chao Yang. 2024. Centauri: Enabling Efficient Scheduling for Communication-Computation Overlap in Large Model Training via Communication Partitioning. In *Proceedings of the 29th ACM International Conference on Architectural Support for Programming Languages and Operating Systems, Volume 3*. ACM, La Jolla CA USA, 178–191. doi:10.1145/3620666.3651379

[3] Yihua Cheng, Yuhan Liu, Jiayi Yao, Yuwei An, Xiaokun Chen, Shaoting Feng, Yuyang Huang, Samuel Shen, Kuntai Du, and Junchen Jiang. 2025. LMCache: An Efficient KV Cache Layer for Enterprise-Scale

LLM Inference. *arXiv preprint arXiv:2510.09665* (2025).

[4] Daniele De Sensi, Lorenzo Pichetti, Flavio Vella, Tiziano De Matteis, Zebin Ren, Luigi Fusco, Matteo Turisini, Daniele Cesarini, Kurt Lust, Animesh Trivedi, Duncan Roweth, Filippo Spiga, Salvatore Di Girolamo, and Torsten Hoefler. 2024. Exploring GPU-to-GPU Communication: Insights into Supercomputer Interconnects. In *SC24: International Conference for High Performance Computing, Networking, Storage and Analysis*. IEEE, Atlanta, GA, USA, 1–15. doi:10.1109/SC41406.2024.00039

[5] Jiangfei Duan, Runyu Lu, Haojie Duanmu, Xiuhong Li, Xingcheng Zhang, Dahua Lin, Ion Stoica, and Hao Zhang. 2024. MuxServe: flexible spatial-temporal multiplexing for multiple LLM serving. In *Proceedings of the 41st International Conference on Machine Learning* (Vienna, Austria) *(ICML'24)*. JMLR.org, Article 473, 13 pages.

[6] Paul Elvinger, Foteini Strati, Natalie Enright Jerger, and Ana Klimovic. 2026. Understanding GPU Resource Interference One Level Deeper. In *Proceedings of the 2025 ACM Symposium on Cloud Computing (SoCC '25)*. Association for Computing Machinery, New York, NY, USA, 687–694. doi:10.1145/3772052.3772270

[7] Shiwei Gao, Qing Wang, Shaoxun Zeng, Youyou Lu, and Jiwu Shu. 2025. Weaver: Efficient {Multi-LLM} Serving with Attention Offloading. In *2025 USENIX Annual Technical Conference (USENIX ATC 25)*. 587–595.

[8] Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. 2016. Deep residual learning for image recognition. In *Proceedings of the IEEE conference on computer vision and pattern recognition*. 770–778.

[9] Ke Hong, Xiuhong Li, Minxu Liu, Qiuli Mao, Tianqi Wu, Zixiao Huang, Lufang Chen, Zhong Wang, Yichong Zhang, Zhenhua Zhu, Guohao Dai, and Yu Wang. 2026. Efficient and Adaptable Overlapping for Computation and Communication via Signaling and Reordering. In *Proceedings of the European Conference on Computer Systems (EuroSys)*.

[10] Bobo Huang, Li Jin, Zhihui Lu, Ming Yan, Jie Wu, Patrick CK Hung, and Qifeng Tang. 2019. RDMA-driven MongoDB: An approach of RDMA enhanced NoSQL paradigm for large-Scale data processing. *Information Sciences* 502 (2019), 376–393.

[11] Bobo Huang, Li Jin, ZhiHui Lu, Xin Zhou, Jie Wu, Qifeng Tang, and Patrick CK Hung. 2019. BoR: Toward high-performance permissioned blockchain in RDMA-enabled network. *IEEE Transactions on Services Computing* 13, 2 (2019), 301–313.

[12] Yibo Huang, Yukai Huang, Ming Yan, Jiayu Hu, Cunming Liang, Yang Xu, Wenxiong Zou, Yiming Zhang, Rui Zhang, Chunpu Huang, et al. 2022. An ultra-low latency and compatible PCIe interconnect for rack-scale communication. In *Proceedings of the 18th International Conference on emerging Networking EXperiments and Technologies*. 232–244.

[13] Yibo Huang, Yiming Qiu, Daqian Ding, Patrick Tser Jern Kon, Yiwen Zhang, Yuzhou Mao, Archit Bhatnagar, Mosharaf Chowdhury, Srinivas Devadas, Jiarong Xing, et al. 2025. Remote Direct Code Execution. In *Proceedings of the 24th ACM Workshop on Hot Topics in Networks*. 300–307.

[14] Yibo Huang, Yiming Qiu, Yunming Xiao, Archit Bhatnagar, Sylvia Ratnasamy, and Ang Chen. 2025. Exposing rdma nic resources for software-defined scheduling. In *Proceedings of the 9th Asia-Pacific Workshop on Networking*. 1–8.

[15] Yibo Huang, Yiming Qiu, Zhenning Yang, Yi Dai, Dingming Wu, Fan Lai, Jiarong Xing, and Ang Chen. 2025. Towards fully disaggregated recommendation model serving. In *Proceedings of the 16th ACM SIGOPS Asia-Pacific Workshop on Systems*. 38–45.

[16] Changho Hwang, Peng Cheng, Roshan Dathathri, Abhinav Jangda, Saeed Maleki, Madan Musuvathi, Olli Saarikivi, Aashaka Shah, Ziyue Yang, Binyang Li, Caio Rocha, Qinghua Zhou, Mahdieh Ghazimirsaeed, Sreevatsa Anantharamu, and Jithin Jose. 2026. MSCCL++: Rethinking GPU Communication Abstractions for AI Inference. In *Proceedings of the 31st ACM International Conference on Architectural Support for Programming Languages and Operating Systems, Volume 2* (USA) *(ASPLOS '26)*. Association for Computing Machinery, New York, NY, USA, 1201–1215. doi:10.1145/3779212.3790188

[17] Inferact. 2026. codex_swebenchpro_traces. https://huggingface.co/datasets/Inferact/codex_swebenchpro_traces. SWE-bench Pro agentic workload traces collected from Codex agent runs.

[18] Xuanlin Jiang, Yang Zhou, Shiyi Cao, Ion Stoica, and Minlan Yu. 2025. Neo: Saving gpu memory crisis with cpu offloading for online llm inference. *Proceedings of Machine Learning and Systems* 7 (2025).

[19] Xinhao Kong, Yibo Zhu, Huaping Zhou, Zhuo Jiang, Jianxi Ye, Chuanxiong Guo, and Danyang Zhuo. 2022. Collie: Finding performance anomalies in {RDMA} subsystems. In *19th USENIX Symposium on Networked Systems Design and Implementation (NSDI 22)*. 287–305.

[20] 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 the 29th Symposium on Operating Systems Principles*. 611–626.

[21] Jiamin Li, Yimin Jiang, Yibo Zhu, Cong Wang, and Hong Xu. 2023. Accelerating Distributed MoE Training and Inference with Lina. In *2023 USENIX Annual Technical Conference (USENIX ATC 23)*. USENIX Association, Boston, MA, 945–959. https://www.usenix.org/conference/atc23/presentation/li-jiamin

[22] Bowen Liu, Xinyang Huang, Qijing Li, Zhuobin Huang, Yijun Sun, Wenxue Li, Junxue Zhang, Ping Yin, and Kai Chen. 2025. CEIO: A Cache-Efficient Network I/O Architecture for NIC-CPU Data Paths. In *Proceedings of the ACM SIGCOMM 2025 Conference*. 381–394.

[23] Hongyi Liu, Jiarong Xing, Yibo Huang, Danyang Zhuo, Srinivas Devadas, and Ang Chen. 2023. Remote direct memory introspection. In *32nd USENIX Security Symposium (USENIX Security 23)*. 6043–6060.

[24] Yadong Liu, Yunming Xiao, Xuan Zhang, Weizhen Dang, Huihui Liu, Xiang Li, Zekun He, Jilong Wang, Aleksandar Kuzmanovic, Ang Chen, and Congcong Miao. 2025. Unlocking ECMP Programmability for Precise Traffic Control. In *22nd USENIX Symposium on Networked Systems Design and Implementation, NSDI 2025, Philadelphia, PA, USA, April 28-30, 2025*. USENIX Association, 87–106. https://www.usenix.org/conference/nsdi25/presentation/liu-yadong

[25] Ziming Mao, Yihan Zhang, Chihan Cui, Kaichao You, Zhongjie Chen, Zhiying Xu, Scott Shenker, Costin Raiciu, Yang Zhou, and Ion Stoica. 2026. UCCL-EP: Portable Expert-Parallel Communication. *USENIX OSDI* (2026).

[26] Hao Mei, Lizhou Gao, Yuanyi Zhu, Liang Wang, Peng Yang, Chao Pei, Chuhao Chen, Zijian Li, Jian Zhao, Dongbo Gu, Hongchen Ren, Jiyuan Chen, Junpeng Zhang, Yunpeng Guan, Jianye Yuan, Jian Wang, Yibo Huang, and Yang Xu. 2026. DistDPU: A Disaggregated DPU Architecture for High-Performance and Cost-Efficient AI Clouds. In *Proceedings of the ACM SIGCOMM 2026 Conference*. Association for Computing Machinery, New York, NY, USA, 519–534. https://doi.org/10.1145/3789240.3829161

[27] Teng Mei, Chengxuan Pei, Marco Canini, and Yunming Xiao. 2026. MCSched: Memory-Controller-Aware Scheduling for Embodied LLM Workloads on NVIDIA Jetson. In *Proceedings of the 17th ACM SIGOPS*

*Asia-Pacific Workshop on Systems, APSys 2026, Bangkok, Thailand, September 17-18, 2026*. ACM.

[28] Deepak Narayanan, Aaron Harlap, Amar Phanishayee, Vivek Seshadri, Nikhil R. Devanur, Gregory R. Ganger, Phillip B. Gibbons, and Matei Zaharia. 2019. PipeDream: generalized pipeline parallelism for DNN training. In *Proceedings of the 27th ACM Symposium on Operating Systems Principles* (Huntsville, Ontario, Canada) *(SOSP '19)*. Association for Computing Machinery, New York, NY, USA, 1–15. doi:10.1145/3341301.3359646

[29] Kelvin K. W. Ng, Henri Maxime Demoulin, and Vincent Liu. 2023. Paella: Low-latency Model Serving with Software-defined GPU Scheduling. In *Proceedings of the 29th Symposium on Operating Systems Principles* (Koblenz, Germany) *(SOSP '23)*. Association for Computing Machinery, New York, NY, USA, 595–610. doi:10.1145/3600006.3613163

[30] NVIDIA. 2024. NVIDIA H100 Tensor Core GPU Datasheet. https://resources.nvidia.com/en-us-hopper-architecture/nvidia-tensor-core-gpu-datasheet.

[31] NVIDIA. 2025. DGX SuperPOD Architecture. https://docs.nvidia.com/dgx-superpod/reference-architecture-scalable-infrastructure-h100/latest/dgx-superpod-architecture.html.

[32] NVIDIA. 2025. IMEX Service for NVLink Networks: Overview. https://docs.nvidia.com/multi-node-nvlink-systems/imex-guide/overview.html.

[33] Pratyush Patel, Esha Choukse, Chaojie Zhang, Aashaka Shah, Íñigo Goiri, Saeed Maleki, and Ricardo Bianchini. 2025. Splitwise: Efficient Generative LLM Inference Using Phase Splitting. In *Proceedings of the 51st Annual International Symposium on Computer Architecture* (Buenos Aires, Argentina) *(ISCA '24)*. IEEE Press, 118–132. doi:10.1109/ISCA59077.2024.00019

[34] 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 the 23rd USENIX Conference on File and Storage Technologies*. 155–170.

[35] Andre Richter, Christian Herber, Thomas Wild, and Andreas Herkersdorf. 2016. Resolving Performance Interference in SR-IOV Setups with PCIe Quality-of-Service Extensions. In *2016 Euromicro Conference on Digital System Design (DSD)*. 454–462. doi:10.1109/DSD.2016.41

[36] Mohammad Shoeybi, Mostofa Patwary, Raul Puri, Patrick LeGresley, Jared Casper, and Bryan Catanzaro. 2019. Megatron-LM: Training Multi-Billion Parameter Language Models Using Model Parallelism. *arXiv preprint arXiv:1909.08053* (2019).

[37] Xinyi Wan, Penghui Qi, Guangxing Huang, Min Lin, and Jialin Li. 2025. PipeOffload: Improving Scalability of Pipeline Parallelism with Memory Optimization. In *Proceedings of the 42nd International Conference on Machine Learning (Proceedings of Machine Learning Research, Vol. 267)*, Aarti Singh, Maryam Fazel, Daniel Hsu, Simon Lacoste-Julien, Felix Berkenkamp, Tegan Maharaj, Kiri Wagstaff, and Jerry Zhu (Eds.). PMLR, 61942–61955. https://proceedings.mlr.press/v267/wan25c.html

[38] Yongtong Wu, Shaoyuan Chen, Yinmin Zhong, Rilin Huang, Yixuan Tan, Wentao Zhang, Liyue Zhang, Shangyan Zhou, Yuxuan Liu, Shunfeng Zhou, et al. 2026. DualPath: Breaking the Storage Bandwidth Bottleneck in Agentic LLM Inference. *arXiv preprint arXiv:2602.21548* (2026).

[39] Yunming Xiao, Diman Zad Tootaghaj, Aditya Dhakal, Lianjie Cao, Puneet Sharma, and Aleksandar Kuzmanovic. 2024. Conspirator: SmartNIC-Aided Control Plane for Distributed ML Workloads. In *Proceedings of the 2024 USENIX Annual Technical Conference, USENIX ATC 2024, Santa Clara, CA, USA, July 10-12, 2024*. USENIX Association, 767–784. https://www.usenix.org/conference/atc24/presentation/xiao

[40] Zhiqiang Xie, Ziyi Xu, Mark Zhao, Yuwei An, Vikram Sharma Mailthody, Scott Mahlke, Michael Garland, and Christos Kozyrakis. 2026. Contextra: Hierarchical Context Caching for Long Context Language Model Serving. *USENIX OSDI* (2026).

[41] Shan Yu, Jiarong Xing, Yifan Qiao, Mingyuan Ma, Yangmin Li, Yang Wang, Shuo Yang, Zhiqiang Xie, Shiyi Cao, Ke Bao, Ion Stoica, Harry Xu, and Ying Sheng. 2025. Prism: Unleashing GPU Sharing for Cost-Efficient Multi-LLM Serving. *arXiv preprint arXiv:2505.04021* (2025). arXiv:2505.04021 [cs.DC] https://arxiv.org/abs/2505.04021

[42] Shulai Zhang, Ningxin Zheng, Haibin Lin, Ziheng Jiang, Wenlei Bao, Chengquan Jiang, Qi Hou, Weihao Cui, Size Zheng, Li-Wen Chang, et al. 2025. Comet: Fine-grained computation-communication overlapping for mixture-of-experts. *Proceedings of Machine Learning and Systems* 7 (2025).

[43] Chenggang Zhao, Shangyan Zhou, Liyue Zhang, Chengqi Deng, Zhean Xu, Yuxuan Liu, Kuai Yu, Jiashi Li, and Liang Zhao. 2025. DeepEP: an efficient expert-parallel communication library. https://github.com/deepseek-ai/DeepEP.

[44] 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 *18th USENIX Symposium on Operating Systems Design and Implementation (OSDI 24)*. 193–210.

[45] Yang Zhou, Zhongjie Chen, Ziming Mao, ChonLam Lao, Shuo Yang, Pravein Govindan Kannan, Jiaqi Gao, Yilong Zhao, Yongji Wu, Kaichao You, Fengyuan Ren, Zhiying Xu, Costin Raiciu, and Ion Stoica. 2026. UCCL-Tran: An Extensible Software Transport Layer for GPU Networking. *USENIX OSDI* (2026).

[46] Yu Zhu, Aditya Dhakal, Pedro Bruel, Gourav Rattihalli, Yunming Xiao, Johann Lombardi, and Dejan S. Milojicic. 2025. An RDMA-First Object Storage System with SmartNIC Offload. In *Proceedings of the SC '25 Workshops of the International Conference for High Performance Computing, Networking, Storage and Analysis, SC Workshops 2025, St Louis, MO, USA, November 16-21, 2025*. ACM, 1650–1657. doi:10.1145/3731599.3767522

[47] Yu Zhu, Aditya Dhakal, Yunming Xiao, Dejan S. Milojicic, and Gustavo Alonso. 2026. ObjectCache: Layerwise Object-Storage Retrieval for KV Cache Reuse. *CoRR* abs/2605.22850 (2026). doi:10.48550/ARXIV.2605.22850 arXiv:2605.22850