SAC: DISAGGREGATED KV CACHE SYSTEM FOR SPARSE ATTENTION LLMS WITH CXL

Anonymous authors
Paper under double-blind review

ABSTRACT

The scaling of LLMs toward long-context inference has shifted the primary serving system bottleneck from computation to memory capacity. Traditional solutions for dense attention models rely on RDMA-based disaggregated memory pools, which perform coarse-grained fetching of the entire prefix KV cache from remote storage to local memory before decoding. However, this approach is fundamentally inefficient for emerging sparse attention models. While only a small fraction of KV entries are active during decoding, these systems still fetch the full KV cache locally, leading to severe transmission bottlenecks and local memory wastage. To address this, we propose SAC, the first efficient disaggregated KV cache system optimized for sparse attention models. By leveraging the low-latency, cache-line granularity load/store semantics of Compute Express Link (CXL), SAC fetches only the required top-k KV entries on demand during inference. Evaluations on DeepSeek-V3.2 using SGLang show that SAC achieves $2.1\times$ higher throughput, $9.7\times$ lower TTFT, and $1.8\times$ lower TBT compared to RDMA-based baselines, establishing CXL-based disaggregation as the superior infrastructure for emerging sparse attention models.

1 INTRODUCTION

The rapid scaling of LLM parameters and the growing demand for long-context inference shift the bottleneck from computation to memory, where prefix KV cache management has become a critical challenge in LLM serving (Zhou et al., 2024). As local GPU HBM and DRAM fall short of the TB-level requirements of KV cache capacity, RDMA-based disaggregated KV cache systems emerge as the standard solution for memory expansion and cross-node sharing (Qin et al., 2025; Liu et al., 2025b; NVIDIA Corporation, 2025).

In RDMA-based disaggregated systems, RDMA NICs prefetch the entire prefix KV cache into local memory to ensure low-latency access during the subsequent decoding stages of the request. Although this full fetching strategy is optimal for *dense attention* models, it becomes fundamentally inefficient for the emerging generation of *sparse attention* models, such as DeepSeek-V3.2 (Liu et al., 2025a), GLM-5.3 (Z.ai, 2026b) and DeepSeek-V4 (DeepSeek-AI, 2026). The core problem is that only a small fraction of KV entries are active at any decoding step, while the KV cache for the full context must be fetched and remain in local memory during the request. We find that the KV cache full fetching approach directly impacts the efficiency of disaggregated serving systems for sparse attention models due to two primary problems.

(P1) Transmission Bottleneck. Due to the reduced computational complexity, the serving throughput of sparse attention models is typically bounded by the achievable batch size, rather than the compute capacity (Xie et al., 2026). For long-context requests, KV cache size can reach dozens of GBs (Sheng et al., 2023). Maintaining high concurrency in sparse attention serving places substantial pressure on KV cache transmission, including RDMA transfers and memory layout rearrangement, which can cause long queuing delays and degrade time to first token and overall throughput.

(P2) Local Memory Wasting. In sparse attention architectures, only the top-k KV entries are utilized per layer for attention calculation. We observe that during the decoding of a long-context request, only a negligible portion of the prefix KV cache is actually accessed. However, the entire prefix KV cache must still be fetched into local memory. To prevent local memory capacity from limiting the

Reviewers: please read the Reviewer Guidelines (iclr.cc/Conferences/2027/ReviewerGuidelines) and the AI Policy for Reviewers (iclr.cc/Conferences/2027/AIPolicyForReviewers). If you used AI to expand, edit, or polish your review, please provide the input text to the LLM. Better still, consider skipping the LLM and submitting your original text: we, and the authors, are much more interested in your unedited thoughts than in what an LLM has to say. AI-assisted or not, you are putting your name and reputation behind your review: LLM-generated falsehoods, hallucinations or misrepresentations are subject to disciplinary action, which may include desk-rejecting all papers you have authored.

request batch size, TB-level memory is required to maintain high concurrency in sparse attention model serving, resulting in high infrastructure costs and poor memory efficiency.

An intuitive approach to solve the issue is to transmit only the top-k KV cache entries on demand, which would reduce transmission pressure and conserve local memory. However, this strategy is infeasible for RDMA-based disaggregated systems. First, the top-k indices are dynamically determined per layer at runtime based on the current query vector. The process of fetching the top-k KV is highly latency-sensitive, and the inherent access latency of RDMA cannot meet the strict timing requirement (Zhu et al., 2024). Second, the sparse top-k KV entries consist of discrete, small data segments. Fetching these data via RDMA requires dozens of independent requests or complex gather/scatter operations, which incur significant software stack overhead (Zhong et al., 2024a).

Fortunately, the emerging **Compute Express Link (CXL)** technology (Das Sharma et al., 2024) introduces new opportunities for disaggregated KV cache systems in sparse attention models. CXL is built on PCIe physical layer with a streamlined protocol stack, offering significantly lower latency than RDMA. Moreover, CXL supports load/store semantics at cache-line granularity with zero message protocol overhead, making it uniquely suited for reading sparse and fine-grained data. Leveraging these features, we explore the feasibility of using CXL-based disaggregated memory to overcome the limitations of RDMA in sparse attention model serving.

We present **SAC** (Sparse Attention on CXL), which stores the KV cache in disaggregated CXL memory accessible to multiple servers. Due to its near-DRAM access latency and support for fine-grained load/store semantics, SAC enables the real-time fetching of top-k KV entries during sparse attention calculations without significantly compromising serving performance. By transferring only the required top-k entries on demand, SAC eliminates the transmission bottleneck (P1). Furthermore, because the full KV cache resides in the disaggregated CXL pool at all times, local memory is freed from maintaining redundant data, thereby resolving the local memory wasting problem (P2).

We integrate SAC into SGLang and evaluate on DeepSeek-V3.2 end-to-end serving across context lengths from 16K to 128K tokens. Compared with RDMA pool, SAC improves the decoding instance performance by achieving 2.1× higher throughput, 9.7× lower time to first token (TTFT), and 1.8× lower time between token (TBT). Compared with non-disaggregated upper bounds, SAC only incurs a 9% throughput degradation, demonstrating the architectural superiority of CXL.

In summary, we make the following contributions:

- **Bottleneck Analysis.** We provide the first systematic analysis of the limitations inherent in traditional RDMA-based KV cache pooling for sparse attention models.
- **SAC System Design.** We design and implement the first CXL-based disaggregated KV cache system optimized for the serving of sparse attention models.
- **End-to-end Evaluation.** We integrate SAC into SGLang and evaluate using DeepSeek-V3.2. Our results demonstrate SAC significantly outperforms RDMA-based baselines, positioning CXL-based disaggregation as superior infrastructure for emerging sparse attention models.

2 Background

2.1 Sparse Attention in LLMs

Sparse attention is a recent advancement in Transformer architectures that reduces quadratic computational complexity to near-linear by restricting token attention to a relevant subset of the sequence. Recent LLMs have transitioned from training-free sparsity strategies (Tang et al., 2024; Zhang et al., 2023; Zhao et al., 2024; Li et al., 2024; Xiao et al., 2023; Liu et al., 2024) toward native sparse attention architectures, such as DeepSeek-V3.2 (Liu et al., 2025a), GLM-5.3 (Z.ai, 2026b), and DeepSeek-V4 (DeepSeek-AI, 2026), demonstrating the robustness and practicality of sparse attention for large-scale deployment. The core mechanism driving this shift is *DeepSeek Sparse Attention* (DSA) (Liu et al., 2025a).

As illustrated in Figure 1, DSA employs a lightweight *Lightning Indexer* to dynamically compute relevance scores using low-dimensional projections of latent KV vectors and queries. These projected

indexer keys are computed once for each historical KV entry and stored in GPU memory for low-latency access. Based on these scores, a fine-grained selection mechanism retrieves only the top-k (2048 in the current DSA implementation) KV latent vectors for Multi-head Latent Attention (MLA) calculations. Consequently, the computational cost of attention is reduced from $O(L^2)$ to $O(kL)$, enabling models to handle significantly longer input sequences and achieve higher serving throughput without compromising accuracy.

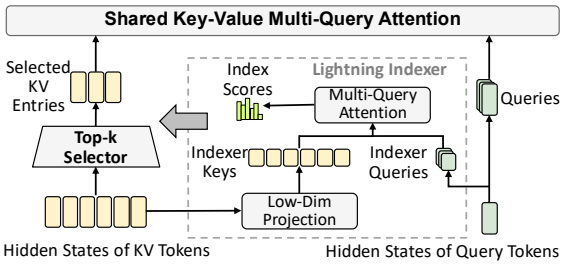

Figure 1: Workflow of DeepSeek Sparse Attention.

2.2 Disaggregated Memory in LLM Serving

The rapid scaling of LLM parameters and the escalating demand for long-context inference have made memory a critical bottleneck in GPU-accelerated serving systems, especially for the storage of KV cache. To address this, the industry is shifting toward RDMA-based disaggregated memory in LLM serving system to achieve memory capacity expansion and memory sharing across nodes (Chen et al., 2024), as shown in Figure 2(a). Mainstream solutions, such as MoonCake (Qin et al., 2025) and LMCache (Liu et al., 2025b), use a CPU-driven RDMA access model to fetch remote data. The host CPU orchestrates the data transfers by moving the data from the GPU into an intermediate bounce buffer located in the host DRAM before finally issuing the RDMA request to send it to the remote pool.

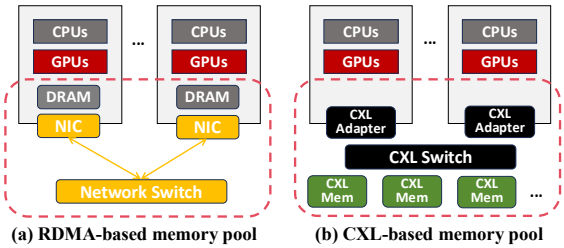

Figure 2: RDMA and CXL disaggregated memory.

While RDMA pools successfully extend memory capacity and facilitate prefix sharing across multi-server clusters, they exhibit inherent bottleneck when storing the KV cache for sparse attention models. RDMA transfers involve complex protocol stack including memory pinning, queue pair synchronization, context switching among others, leading to a high latency of several microseconds for each transfer (Kalia et al., 2019). Besides, the message-based send/recv semantics lead to significant performance degradation during small packet transfers (Yang et al., 2026). For sparse KV access, each layer needs a small, independently chosen subset of KV entries. The accumulated overhead makes per-layer RDMA reads unacceptable for real-time top-k fetching.

2.3 Compute Express Link (CXL)

CXL is an emerging interconnect protocol built on the PCIe physical layer that facilitates high-bandwidth, low-latency memory sharing between hosts and accelerators (Das Sharma et al., 2024). CXL supports memory expansion through Type-3 memory devices via the CXL.mem protocol. Modern CXL 2.0/3.0 switches (XConn Technologies, 2023) further enable multi-node memory sharing, providing a scalable hardware foundation for managing memory-intensive workloads. As illustrated in Figure 2(b), CXL simplifies the data path compared to traditional RDMA by providing a hardware-managed load/store memory semantic. Cache-line granularity makes CXL uniquely suited for fine-grained, sparse memory lookups, with remote access latencies approaching local DRAM.

Consequently, CXL has emerged as a compelling alternative to RDMA for memory pooling across diverse data center workloads, including databases (Yang et al., 2025; Huang et al., 2025; Wang et al., 2024) and cloud platforms (Zhong et al., 2024b; Guan et al., 2025; Li et al., 2023). Recent research also highlights its potential in LLM, particularly for KV cache management. Beluga (Yang et al., 2026) and TRACT (Yoon et al., 2025) manage dense models' prefix KV cache in disaggregated CXL memory, prefetching the entire prefix into local memory like RDMA. Sparse attention instead requires per-layer, on-demand sparse KV reads. The next section analyzes the shortcomings of RDMA-based disaggregation and demonstrates CXL's superiority.

3 MOTIVATION

3.1 BOTTLENECKS OF RDMA-BASED KV CACHE SYSTEM

Current RDMA-based disaggregated systems rely on KV cache full-fetching before decoding, which introduces two fundamental problems that hinder the serving efficiency of sparse attention models.

P1: Transmission Bottleneck. The transmission of massive KV cache blocks imposes significant pressure on RDMA bandwidth. Since a request cannot be processed until its entire prefix KV cache is retrieved, high-concurrency workloads frequently trigger network contention and substantial queuing delays. Additionally, KV caches are typically stored using a page-first layout on the remote but require a layer-first layout for local processing. This layout transformation introduces additional latency overhead. Figure 3 illustrates the average latency for KV cache transmission under 100 Gbps RNIC. The latency grows near-linearly with concurrency and context length. In high-concurrency scenarios, the latency reach dozens of seconds, directly degrading the TTFT. When transmission becomes the bottleneck, the system cannot retrieve enough requests for computation, limiting the serving throughput.

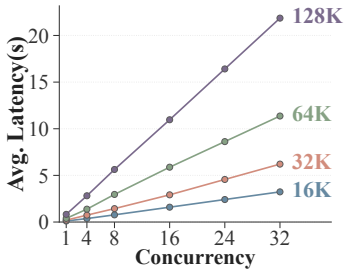

Figure 3: RDMA prefetch latency for prefix KV cache.

P2: Local Memory Wasting. Fetching the full KV cache locally via RDMA is fundamentally inefficient for sparse attention models, as only a small fraction of the KV cache is active during decoding. Figure 4 profiles DeepSeek-V3.2 serving requests from the ShareGPT dataset (Chiang et al., 2023) with 1K-token output, the unused portion of the KV cache grows alongside context length. In 128K-context scenarios, only 21% of the KV cache is actually utilized. Long-context KV caches also have a substantial footprint: DeepSeek-V3.2 requires 9.2 GB per request at 128K context, so high concurrency requires TB-level local memory. Thus, full-fetching is unnecessary for sparse attention and imposes an excessive local memory burden.

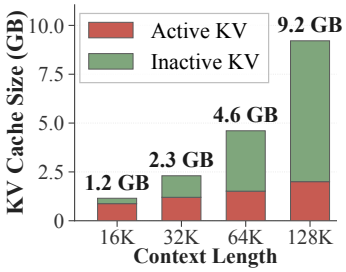

Figure 4: KV cache usage and footprint for DeepSeek-V3.2.

3.2 MEMORY ACCESS LATENCY OF SPARSE ATTENTION KV CACHE

Sparse attention serving requires efficient, real-time loading of per-layer top-k KV entries from disaggregated memory. We profile sparse KV retrieval latency by randomly sampling indices from a 128K-token context. Each DeepSeek-V3.2 MLA KV entry contains a 512-dimensional latent KV vector and a 64-dimensional RoPE vector in `bfloat16`. We measure GPU read latency across (i) disaggregated CXL memory, (ii) disaggregated RDMA memory, and (iii) local DRAM as baseline.

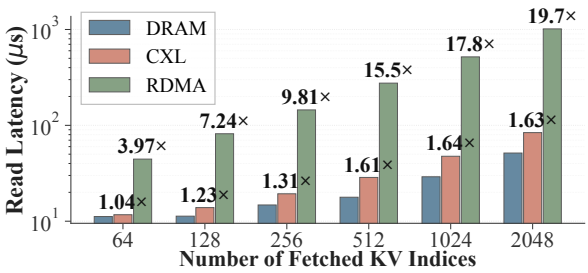

Figure 5: Retrieval latency comparison for sparse KV.

Figure 5 illustrates the mean fetch latencies for varying numbers of sparse KV entries. CXL-based disaggregated memory achieves near-DRAM performance, maintaining a latency within 1.04–$1.64\times$ of local DRAM. This demonstrates that CXL is an ideal fabric for sparse attention models. In contrast, RDMA exhibits significantly higher latency, ranging from 4.0–$19.7\times$ slower than local DRAM and reaching millisecond-level delays as the number of KV entries increases. These latencies make per-layer top-k KV fetching via RDMA impractical for inference, severely degrading performance. We therefore exclude dynamic RDMA top-k fetching from subsequent experiments and compare CXL against RDMA full-prefetch to demonstrate CXL's architectural superiority.

4 System Design

4.1 SAC Workflow

SAC transitions KV cache management for sparse attention models from RDMA to a CXL-based disaggregated architecture designed for efficient, high-concurrency serving. The system architecture comprises three primary components: the Prefill Instance, the Decode Instance, and the CXL-based Disaggregated KV Cache System, as shown in Figure 6.

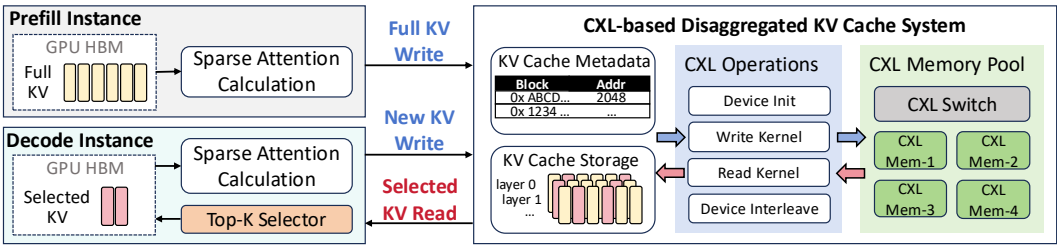

Figure 6: Overview of the SAC system and operational workflow, illustrating the interaction between compute instances and the disaggregated CXL memory pool.

Prefill Instance. The prefill stage is characterized by large batch sizes and is primarily compute-bound. During prefilling, full KV caches are maintained in local GPU memory to ensure fast access. The prefill instance is responsible for populating the CXL memory pool with the calculated KV entries upon completion. Writing full KV blocks to CXL memory is well-established (Yang et al., 2026; Yoon et al., 2025) and not unique to sparse attention models, so it is not our focus.

Decode Instance. The decode instance is built upon HiSparse framework in SGLang (Xie et al., 2026) (detailed in Appendix C), which is designed for high-throughput inference for sparse attention models. It offloads the KV cache to low-tiered memory and uses swap-in kernels to dynamically load only the top-k KV into GPU for attention calculations. SAC extends this framework by integrating disaggregated memory backend and CXL-based KV cache management system. During each attention layer, SAC directly fetches the required top-k KV from CXL memory pool into GPU. The KV entries for newly generated tokens are written back to the CXL memory.

CXL-based Disaggregated KV cache System. The CXL-based system manages the KV cache and its associated metadata, mapping all data into a global CXL address space accessible by multiple compute instances. It handles CXL device initialization and provides optimized read/write kernels and device interleaving strategy. The physical memory pool consists of multiple CXL devices interconnected via a CXL switch, which provides fine-grained, low-latency access required to support the sparse read and write patterns inherent in sparse attention inference.

4.2 SAC System Topology

Figure 7 shows SAC's CXL-based disaggregated architecture, which provides fine-grained, low-latency access for sparse attention. Each server contains host CPUs and multiple GPUs. Each NUMA node connects to the CXL switch through a PCIe 5.0 x16 PCIe/CXL adapter. The CXL memory pool consists of a switch node, which is equipped with XConn XC50256 CXL switch chip (XConn Technologies, 2023). The chip has 256 PCIe 5.0 lanes which are partitioned between CXL memory devices and compute servers. The CXL switch chip provides 512 GB/s of total bandwidth between memory devices and compute servers, which connects up to 8 servers to the CXL memory pool. Hardware-level address mapping and forwarding enable robust multi-host scaling: multiple GPUs retrieve non-contiguous top-k entries with near-DRAM latency, bypassing RDMA message-passing overhead.

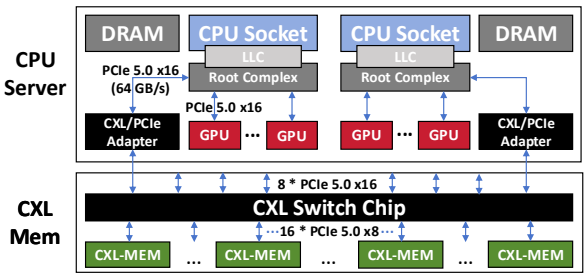

Figure 7: Hardware topology of SAC system.

4.3 CXL-BASED KV CACHE MANAGEMENT

4.3.1 UNIFIED CXL MEMORY RESOURCE

The CXL disaggregated memory serves as a unified resource for both the KV cache data and the metadata index. Compared to RDMA-based architectures, CXL approach offers two advantages.

Locality-Transparent KV Cache Layout. Traditional RDMA-based disaggregated systems suffer from a significant performance gap between local and remote memory (Qin et al., 2025; NVIDIA Corporation, 2025), necessitating complex and cache-aware scheduling policies. In contrast, SAC leverages CXL to achieve near-DRAM latency. Since CXL supports byte-addressable, fine-grained load/store semantics, we adopt a locality-transparent, layer-first memory layout that is identical to the structure used in local GPU. This allows GPU to access KV cache in CXL memory using the same logic as local memory. For sparse attention models, per-layer top-k retrieval is transformed into lightweight memory read operations in the GPU kernels, bypassing complex buffer management and reducing scheduling complexity.

CXL-based Metadata Management. Conventional shared KV cache systems typically rely on centralized metadata services accessed via high-overhead protocols such as RDMA or TCP/IP (Yang et al., 2026). SAC replaces these expensive network-based interactions by hosting metadata within a dedicated, globally accessible CXL shared-memory region. By utilizing native load/store semantics for cross-server synchronization, SAC replaces heavy remote procedure call (RPC) interactions with high-speed memory operations. This approach ensures that metadata updates and lookups are performed with near-DRAM latency and eliminates kernel-mode transitions and context switches, providing a responsive control plane for high-concurrency sparse attention workloads.

4.3.2 CXL OPERATION IMPLEMENTATION

CXL Device Initialization. CXL devices are exposed as DAX (Direct Access) devices in operating system. A `mmap` system call maps the DAX device to the host virtual address. With `cudaHostRegister`, GPU accesses CXL using the same load/store instructions as DRAM.

GPU Write Path. The prefill instance transfers the KV cache in GPU to CXL memory through a layer-wise offloading strategy, which is executed by a memory-coalesced kernel where each warp is assigned a designated subset of KV indices. By using vectorized store instructions (`st.global.b64`), the kernel achieves high-bandwidth, single-pass writes into the CXL global address space.

GPU Read Path. The decoding instance retrieves the top-k KV required for each attention layer from the CXL memory. The read path is optimized for sparse, non-contiguous access to multiple fixed-length vectors. With a coalesced kernel, SAC uses vectorized load instructions (`ld.global.b64`) to efficiently fetch the data into GPU, satisfying the real-time constraints of sparse attention inference.

4.3.3 CXL BANDWIDTH OPTIMIZATION

The CXL pool contains multiple devices, each connected to the switch via PCIe 5.0 x8. Optimizing upstream bandwidth requires balancing traffic across devices. For MLA-based sparse attention, data parallel attention avoids replicated KV cache storage by computing each request's attention on one GPU. The challenge is to reduce contention when multiple GPUs access the same CXL device concurrently.

SAC stores each request's KV cache on one CXL device. When dispatching requests to model runners, the scheduler accounts for their placement so GPUs access different devices in an interleaved manner. We assign GPU ranks to CXL devices in a round-robin fashion, as shown in Figure 8. By distributing traffic across multiple physical links during parallel GPU access, this approach alleviates single-link pressure and significantly reduces retrieval latency.

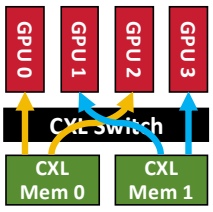

Figure 8: CXL interleaving example.

5 EVALUATION

We integrate SAC into SGLang (Zheng et al., 2024) and HiSparse framework (Xie et al., 2026). We evaluate the DeepSeek-V3.2 model (Liu et al., 2025a) (AWQ 4-bit quantization) using an 8-H20 GPU server and 2TB CXL memory pool as KV cache backend. We compare SAC against two alternative backends: RDMA-based memory pool and local host DRAM as upper-bound baseline. Detailed experimental setup is provided in Appendix A.

5.1 END-TO-END PERFORMANCE COMPARISON

We evaluate the end-to-end performance of SAC across three memory backends. To isolate the performance impact on the prefill and decoding stages, we design a two-round experimental setup:

- **Round-1 (Cache Populate):** This scenario evaluates the prefill instance. The initial KV cache is computed on the GPU and subsequently stored in the disaggregated memory pool.
- **Round-2 (Cache Hit)**: This scenario evaluates the decode instance, where all the KV cache is pre-populated in the memory pool and the prefill stage is bypassed entirely.

We measure serving metrics including output token throughput, time between tokens (TBT) and time to first token (TTFT). The benchmark uses 512 requests sampled from the ShareGPT dataset (Chiang et al., 2023), with concurrency set to 8 for Round-1 and 64 for Round-2. We sweep the context length from 16K to 128K tokens and fix the output length at 1K tokens. Additional results with varying output lengths are provided in Appendix D.2.

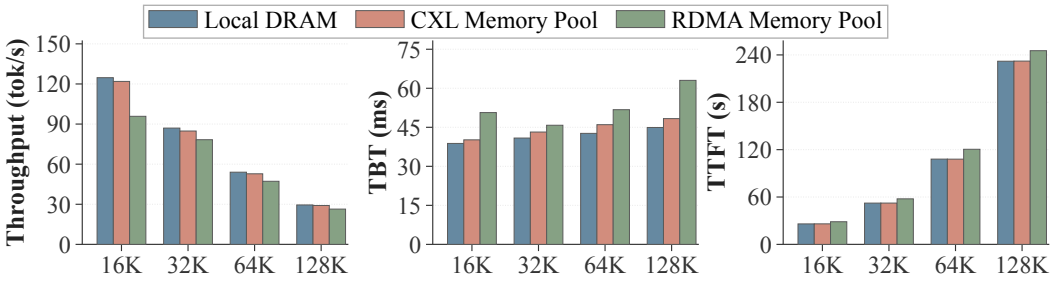

Figure 9: Round-1 performance comparison of SAC against RDMA and local DRAM backend.

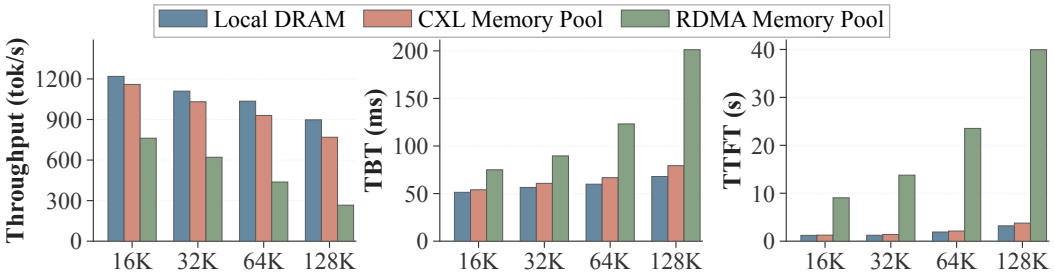

Figure 10: Round-2 performance comparison of SAC against RDMA and local DRAM backend.

As illustrated in Figure 9 and 10, SAC closely approximates the performance of the local DRAM upper bound in both Round-1 and Round-2. On average, SAC achieves a throughput within 91% of the DRAM baseline, with only a marginal increase in TBT and TTFT. In Round-1 prefill stage, since both CXL and RDMA maintain the full KV cache on the GPU for computation and subsequently store the KV cache into the memory pool, the two backends exhibit comparable performance. In Round-2, SAC significantly outperforms RDMA baseline, delivering on average $2.1\times$ higher throughput. RDMA suffers from a substantially higher $9.7\times$ TTFT than SAC, as high client concurrency saturates the transmission bandwidth during KV retrieval, resulting in increased queuing delays. Besides, the TBT of RDMA is $1.8\times$ higher than SAC due to severe PCIe bus contention, which must simultaneously handle incoming KV cache traffic and the HiSparse swap-in process.

5.2 Throughput Scalability Comparison

We evaluate the decoding throughput across varying concurrency to demonstrate the scalability of SAC and the transmission bottleneck inherent in RDMA-based systems. We compare SAC with RDMA backend at context lengths of 32K, 64K, and 128K.

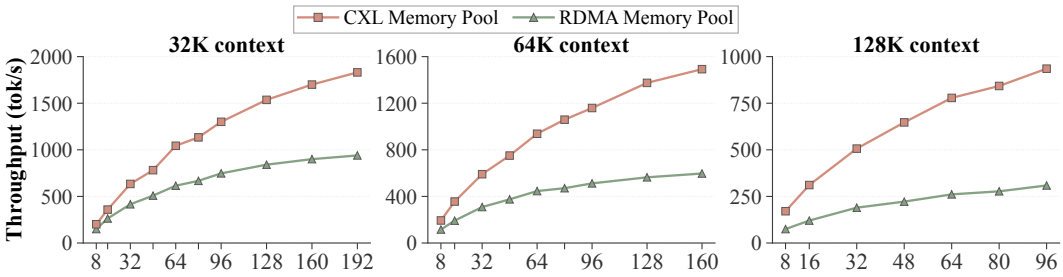

Figure 11: Decoding throughput scalability comparison of SAC and RDMA under concurrency.

As illustrated in Figure 11, SAC exhibits robust throughput scalability, with performance increasing consistently alongside concurrency. For 32K, 64K, and 128K, SAC achieves up to $2.0\times$, $2.5\times$ and $3.1\times$ higher throughput than the RDMA-based system. In contrast, RDMA backend quickly reaches a throughput plateau due to the full fetching of KV cache, which exhausts the transmission bandwidth. SAC bypasses this bottleneck by leveraging CXL fine-grained load/store semantics to fetch only the necessary top-k entries. This significantly reduces the per-request bandwidth footprint, allowing SAC to sustain high-throughput decoding even as context length and concurrency scale.

5.3 Comparison with Non-Disaggregated Baselines

We compare the decoding throughput of SAC against non-disaggregated configurations to demonstrate the architectural superiority of CXL memory. We evaluate SAC against local DRAM and GPU-only baselines, where KV cache is stored in host memory and GPU HBM respectively.

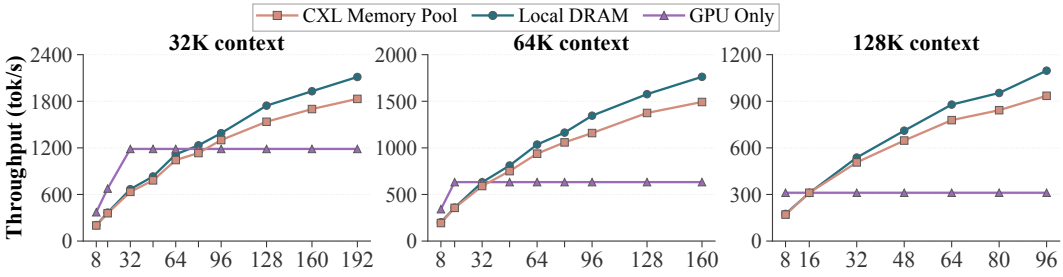

Figure 12: Throughput comparison of SAC with non-disaggregated baselines under concurrency.

As illustrated in Figure 12, although SAC stores the KV cache in disaggregated CXL memory, it achieves performance close to the local DRAM baseline. When compared to the HBM-only baseline, the GPU HBM delivers higher throughput at low concurrency. However, as concurrency increases, the HBM reaches a capacity limit where the maximum achievable batch size cannot increase further. This demonstrates the importance of using lower-tier memory for sparse attention inference, which provides the necessary capacity to scale to high-concurrency workloads.

5.4 Impact of CXL Device Interleaving

We conduct an ablation study to evaluate the effectiveness of device-aware interleaving strategy introduced in Section 4.3.3. We compare the decoding throughput using a single CXL memory device against an interleaved configuration across two CXL devices. As illustrated in Figure 13, the interleaved configuration consistently outperforms the single-device baseline. On average, interleaving improves decoding throughput by 9.2%, with a peak gain of 14.2% at 128K context

length. This demonstrates that by interleaving KV cache storage across multiple devices, SAC effectively mitigates link-level contention. The results suggest that scaling to additional CXL devices would further bridge the performance gap between SAC and the local DRAM baseline by providing greater aggregate bandwidth to sustain high-performance inference for long-context workloads.

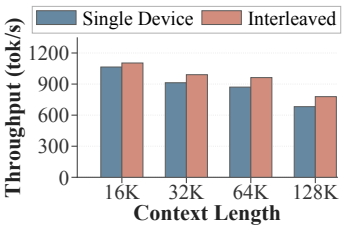

Figure 13: Throughput gain from CXL device interleaving.

5.5 Impact of HiSparse Configuration

SAC is built upon HiSparse, where a critical configuration parameter is `device_buffer_size`. This parameter determines the capacity of the hot KV cache stored in GPU HBM, which directly impacts the volume of KV cache data transmitted from disaggregated memory to the GPU. We evaluate two configurations: `device_buffer_size` set to 4K and 6K.

As illustrated in Figure 14, SAC performance improves as the buffer size increases; the 6K configuration achieves an average throughput 10.4% higher than the 4K baseline. This improvement is driven by a lower KV cache miss rate, which reduces both the total transmission data size and the resulting pressure on the CXL link. These results suggest that maintaining a larger GPU buffer is beneficial for SAC performance, as it minimizes the overhead of data transfers from the disaggregated memory.

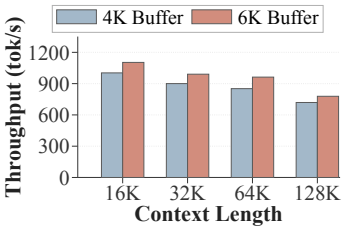

Figure 14: Impact of GPU device buffer size.

6 Discussion

Generality to Other Models. DeepSeek-V4 (DeepSeek-AI, 2026) combines compressed sparse attention with heavily compressed attention, retaining top-k KV retrieval in the sparse component. GLM-5.2 and GLM-5.3 (Z.ai, 2026a;b) retain the core DSA architecture of DeepSeek-V3.2 and incorporate cross-layer indexer sharing. These models preserve indexer-guided sparse KV access, making the fine-grained retrieval mechanism in SAC applicable through CXL load/store semantics.

Memory Pool Paradigm Transform. The evolution of model sparsity is driving KV cache access toward a fine-grained, heterogeneous pattern. Traditional message-based protocols cannot satisfy the requirements. LLM serving clusters are evolving toward memory-semantic interconnects, necessitating tighter coupling between compute and memory nodes to maximize performance. CXL serves as a cornerstone for this transition and the emergence of unified scale-up protocols (Liao et al., 2025; Arsid, 2025) promises more robust solutions for high-performance, disaggregated KV cache storage.

Limitation and Future Work. The end-to-end hardware evaluation focuses on DeepSeek-V3.2; evaluation on recent models, including GLM-5.3 and DeepSeek-V4, remains future work. Experimental platform limitations prevent hardware experiments with multiple compute nodes sharing a centralized CXL memory pool. Appendix E therefore presents a simulation case study of high-throughput rollouts, demonstrating how shared KV residency and sparse retrieval reduce full P-to-D handoffs and alleviate receive-bandwidth bottlenecks at decode nodes.

7 Conclusion

We address the fundamental inefficiency of traditional RDMA-based KV cache systems for emerging sparse attention models. We identify that the KV cache full-fetching in RDMA-based systems leads to transmission bottlenecks and local memory waste. To overcome the limitations, we present SAC, a disaggregated KV cache system that leverages the low-latency, fine-grained load/store semantics of CXL. By fetching only the required top-k KV entries on demand, SAC effectively eliminates the transmission overhead and enables high-concurrency serving without the burden of TB-level local memory. Our end-to-end evaluation with DeepSeek-V3.2 demonstrates that SAC achieves 2.1$\times$ higher throughput, 9.7$\times$ lower TTFT, and 1.8$\times$ lower TBT compared to RDMA-based baselines. As LLMs evolve toward higher sparsity and longer contexts, SAC demonstrates that CXL-based disaggregation provides a superior foundation for scalable, efficient next-generation LLM serving.

AI USE STATEMENT

We used generative AI tools to assist with writing code. The main text was written by the authors, with generative AI tools used to polish its language. For the appendix, the authors prepared the outline, generative AI tools assisted with drafting, and the authors proofread and revised the resulting text. The authors take responsibility for the final manuscript and accompanying code, including content produced with the assistance of generative AI.

REFERENCES

Minwoo An, Jinho Kim, Juhyun Oh, Myung-Jae Lee, and Myoungsoo Jung. Cxl-dl: A cxl-based shared memory architecture for deep learning. In *Proceedings of the 30th IEEE International Symposium on High-Performance Computer Architecture (HPCA)*, pp. 1–14, 2024.

Rajesh Arsid. Ultra ethernet and ualink: Next-generation interconnects for ai infrastructure. *IJSAT-International Journal on Science and Technology*, 16(2), 2025.

Shiyang Chen, Rain Jiang, Dezhi Yu, Jinlai Xu, Mengyuan Chao, Fanlong Meng, Chenyu Jiang, Wei Xu, and Hang Liu. KVDirect: Distributed disaggregated LLM inference. *arXiv preprint arXiv:2501.14743*, 2024.

Wei-Lin Chiang, Zhuohan Li, Zi Lin, Ying Sheng, Zhanghao Wu, Hao Zhang, Lianmin Zheng, Siyuan Zhuang, Yonghao Zhuang, Joseph E. Gonzalez, Ion Stoica, and Haier Duan. Vicuna: An open-source chatbot impressing gpt-4 with 90%* chatgpt quality, March 2023. URL `https://lmsys.org/blog/2023-03-30-vicuna/`.

Counterpoint Research. Memory monthly pricing tracker, april 2026: LTA structurally reshaping DRAM pricing dynamics. Technical report, Counterpoint Research, April 2026. URL `https://counterpointresearch.com/en/reports/memory-price-tracker-april-2026`. Published April 7, 2026; public report overview reports finalized Q1 64GB DDR5 RDIMM price of USD 927; accessed September 26, 2026.

Debendra Das Sharma, Robert Blankenship, and Daniel Berger. An introduction to the compute express link (CXL) interconnect. *ACM Computing Surveys*, 56(11):1–37, 2024.

DeepSeek-AI. Deepseek-v4-pro. `https://huggingface.co/deepseek-ai/DeepSeek-V4-Pro`, 2026. Accessed: 2026-05-02.

Tianchan Guan, Yijin Guan, Zhaoyang Du, Jiacheng Ma, Boyu Tian, Zhao Wang, Teng Ma, Zheng Liu, Yang Kong, Yuan Xie, et al. Memtunnel: A CXL-based rack-scale host memory pooling architecture for cloud service. *IEEE Transactions on Parallel and Distributed Systems*, 2025.

Yibo Huang, Haowei Chen, Newton Ni, Yan Sun, Vijay Chidambaram, Dixin Tang, and Emmett Witchel. Tigon: A distributed database for a CXL pod. In *19th USENIX Symposium on Operating Systems Design and Implementation (OSDI 25)*, pp. 109–128, 2025.

Anuj Kalia, Michael Kaminsky, and David Andersen. Datacenter {RPCs} can be general and fast. In *16th USENIX Symposium on Networked Systems Design and Implementation (NSDI 19)*, pp. 1–16, 2019.

Keunsoo Kim, Sang-Won Lee, Jin-Hee Han, Gi-Moon Park, Young-Cheon Kim, and Jung-Hwan Choi. Cxl-pnm: A cxl-based processing-near-memory accelerator for large-scale nlp models. *IEEE Computer Architecture Letters*, 22(2):157–160, 2023.

Huaicheng Li, Daniel S Berger, Lisa Hsu, Daniel Ernst, Pantea Zardoshti, Stanko Novakovic, Monish Shah, Samir Rajadnya, Scott Lee, Ishwar Agarwal, et al. 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*, pp. 574–587, 2023.

Yuhong Li, Yingbing Huang, Bowen Yang, Bharat Venkitesh, Acyr Locatelli, Hanchen Ye, Tianle Cai, Patrick Lewis, and Deming Chen. SnapKV: LLM knows what you are looking for before generation. *Advances in Neural Information Processing Systems*, 37:22947–22970, 2024.

Heng Liao, Bingyang Liu, Xianping Chen, Zhigang Guo, Chuanning Cheng, Jianbing Wang, Xiangyu Chen, Peng Dong, Rui Meng, Wenjie Liu, et al. Ub-mesh: a hierarchically localized nd-fullmesh datacenter network architecture. *IEEE Micro*, 2025.

Aixin Liu, Aoxue Mei, Bangcai Lin, Bing Xue, Bingxuan Wang, Bingzheng Xu, Bochao Wu, Bowei Zhang, Chaofan Lin, Chen Dong, et al. Deepseek-v3. 2: Pushing the frontier of open large language models. *arXiv preprint arXiv:2512.02556*, 2025a.

Guangda Liu, Chengwei Li, Jieru Zhao, Chenqi Zhang, and Minyi Guo. Clusterkv: Manipulating LLM KV cache in semantic space for recallable compression. *arXiv preprint arXiv:2412.03213*, 2024.

Yuhan Liu, Yihua Cheng, Jiayi Yao, Yuwei An, Xiaokun Chen, Shaoting Feng, Yuyang Huang, Samuel Shen, Rui Zhang, Kuntai Du, and Junchen Jiang. LMCache: An efficient KV cache layer for enterprise-scale LLM inference. *arXiv preprint arXiv:2510.09665*, 2025b. URL https://arxiv.org/abs/2510.09665.

Nebius. SWE-agent-trajectories. Hugging Face dataset, 2024. URL https://huggingface.co/datasets/nebius/SWE-agent-trajectories. Accessed September 26, 2026.

NVIDIA. ConnectX-6 VPI Adapter Cards User Manual: Introduction. Hardware documentation, n.d. URL https://networking-docs.nvidia.com/connectx6vpihw/introduction. Single-port 100 Gb/s adapter MCX651105A-EDAT; accessed September 26, 2026.

NVIDIA Corporation. NVIDIA Dynamo Open-Source Library Accelerates and Scales AI Reasoning Models. https://nvidianews.nvidia.com/news/nvidia-dynamoopen-source-library-accelerates-and-scales-ai-reasoningmodels, 2025.

Ruoyu Qin, Zheming Li, Weiran He, Jialei Cui, Feng Ren, Mingxing Zhang, Yongwei Wu, Weimin Zheng, and Xinran Xu. Mooncake: Trading more storage for less computation a KVCache-centric architecture for serving LLM chatbot. In *23rd USENIX Conference on File and Storage Technologies (FAST 25)*, pp. 155–170, 2025.

Router-Switch.com. MCX651105A-EDAT: NVIDIA Mellanox ConnectX-6 VPI 100 gb/s single-port adapter. Product price listing, 2026. URL https://www.router-switch.com/mcx651105a-edat.html. Original-new listing at USD 720; accessed September 26, 2026.

Guangming Sheng, Chi Zhang, Zilingfeng Ye, Xibin Wu, Wang Zhang, Ru Zhang, Yanghua Peng, Haibin Lin, and Chuan Wu. HybridFlow: A flexible and efficient RLHF framework. *arXiv preprint arXiv:2409.19256*, 2024. URL https://arxiv.org/abs/2409.19256.

Ying Sheng, Lianmin Zheng, Binhang Yuan, Zhuohan Li, Max Ryabinin, Beidi Chen, Percy Liang, Christopher Ré, Ion Stoica, and Ce Zhang. FlexGen: High-throughput generative inference of large language models with a single GPU. In *Proceedings of the 40th International Conference on Machine Learning*, volume 202 of *Proceedings of Machine Learning Research*, pp. 31094–31116. PMLR, 2023. URL https://proceedings.mlr.press/v202/sheng23a.html.

Jiaming Tang, Yilong Zhao, Kan Zhu, Guangxuan Xiao, Baris Kasikci, and Song Han. Quest: query-aware sparsity for efficient long-context LLM inference. In *Proceedings of the 41st International Conference on Machine Learning*, pp. 47901–47911, 2024.

Zhonghua Wang, Yixing Guo, Kai Lu, Jiguang Wan, Daohui Wang, Ting Yao, and Huatao Wu. Rcmp: Reconstructing RDMA-based memory disaggregation via CXL. *ACM Transactions on Architecture and Code Optimization*, 21(1):1–26, 2024.

XConn Technologies. XC50256: World's first hybrid CXL 2.0 and PCIe Gen5 switch IC. https://www.xconn-tech.com/product, 2023.

Guangxuan Xiao, Yuandong Tian, Beidi Chen, Song Han, and Mike Lewis. Efficient streaming language models with attention sinks. *arXiv preprint arXiv:2309.17453*, 2023.

Zhiqiang Xie, Zhangheng Huang, Tingwei Huang, Ziyi Xu, Ruiyang Ma, and Christos Kozyrakis. Hisparse: Scaling sparse-attention decoding with hierarchical kv cache management. *arXiv preprint arXiv:2608.07009*, 2026.

Xinjun Yang, Yingqiang Zhang, Hao Chen, Feifei Li, Gerry Fan, Yang Kong, Bo Wang, Jing Fang, Yuhui Wang, Tao Huang, et al. Unlocking the potential of CXL for disaggregated memory in cloud-native databases. In *Companion of the 2025 International Conference on Management of Data*, pp. 689–702, 2025.

Xinjun Yang, Qingda Hu, Junru Li, Feifei Li, Yicong Zhu, Yuqi Zhou, Qiuru Lin, Jian Dai, Yang Kong, Jiayu Zhang, et al. Beluga: A cxl-based memory architecture for scalable and efficient LLM kvcache management. *Proceedings of the ACM on Management of Data*, 4(1 (SIGMOD):1–29, 2026.

Dongha Yoon, Younghoon Min, Hoshik Kim, Sam H Noh, and Jongryool Kim. TraCT: Disaggregated LLM serving with CXL shared memory KV cache at rack-scale. *arXiv preprint arXiv:2512.18194*, 2025.

Z.ai. GLM-5.2: Model card and configuration. Hugging Face model repository, 2026a. URL `https://huggingface.co/zai-org/GLM-5.2`. Accessed September 26, 2026.

Z.ai. GLM-5.3-BF16: Model card. Hugging Face model repository, 2026b. URL `https://huggingface.co/zai-org/GLM-5.3-BF16`. Accessed September 26, 2026.

Zhenyu Zhang, Ying Sheng, Tianyi Zhou, Tianlong Chen, Lianmin Zheng, Ruisi Cai, Zhao Song, Yuandong Tian, Christopher Ré, Clark Barrett, et al. H2o: Heavy-hitter oracle for efficient generative inference of large language models. *Advances in Neural Information Processing Systems*, 36:34661–34710, 2023.

Junqi Zhao, Zhijin Fang, Shu Li, Shaohui Yang, and Shichao He. Buzz: Beehive-structured sparse kv cache with segmented heavy hitters for efficient LLM inference. *arXiv preprint arXiv:2410.23079*, 2024.

Lianmin Zheng, Liangsheng Yin, Zhiqiang Xie, Chuyue Sun, Jeff Huang, Cody H Yu, Shiyi Cao, Christos Kozyrakis, Ion Stoica, Joseph E Gonzalez, et al. Sglang: Efficient execution of structured language model programs. *Advances in neural information processing systems*, 37:62557–62583, 2024.

Yinmin Zhong, Shengyu Liu, Junda Chen, Jian Huang, et al. Distserve: Disaggregating prefill and decoding for goodput-optimized large language model serving. In *Proceedings of the 17th USENIX Symposium on Operating Systems Design and Implementation (OSDI)*, 2024a.

Yuhong Zhong, Daniel S Berger, Carl Waldspurger, Ryan Wee, Ishwar Agarwal, Rajat Agarwal, Frank Hady, Karthik Kumar, Mark D Hill, Mosharaf Chowdhury, et al. Managing memory tiers with CXL in virtualized environments. In *18th USENIX Symposium on Operating Systems Design and Implementation (OSDI 24)*, pp. 37–56, 2024b.

Zixuan Zhou, Xuefei Ning, Ke Hong, Tianyu Fu, Jiaming Xu, Shiyao Li, Yuming Lou, Luning Wang, Zhihang Yuan, Xiuhong Li, et al. A survey on efficient inference for large language models. *arXiv preprint arXiv:2404.14294*, 2024.

Kan Zhu, Wenyi Zhao, et al. Nanoflow: Towards optimal capacity and efficiency in throughput-oriented LLM serving. In *arXiv preprint arXiv:2408.12757*, 2024.

A Experimental Setup

This section provides the detailed hardware and software configuration used in our experiments. To ensure reproducibility, we specify the hardware specifications, memory pool setups, and framework parameters.

A.1 Hardware Platform

All experiments were conducted on a high-performance server equipped with the hardware specifications summarized in Table 1.

Table 1: Hardware Specifications

Component	**Configuration**
Operating System	Ubuntu 22.04 (Kernel 6.2.0-1015)
CPU	2× Intel(R) Xeon(R) Platinum 8575C
L3 Cache	640 MiB (320 MiB per CPU)
System DRAM	2 TB (32× DDR5 4800 MT/s 64 GB)
GPU	8× NVIDIA H20 (96 GB)
Interconnect	4× PCIe Switch (PCIe 5.0)

A.2 Memory Pool Configurations

CXL Memory Pool. The CXL infrastructure is built around an XConn XC50256 CXL switch. We utilize two PCIe 5.0 x16 CXL adapters to interface with the switch. The memory pool uses CXL Type-3 devices. Data Direct I/O (DDIO) is disabled to bypass the Last Level Cache (LLC), ensuring cache coherency across nodes and direct memory access.

RDMA Memory Pool. For the RDMA baseline, we utilize eight ConnectX-7 single-port 100Gbps NICs. We emulate the RDMA pool by storing the KV cache in local DRAM. Data transfers are routed through the RDMA NICs in a loopback configuration to reflect the latency and throughput characteristics of a network-attached memory pool. As physical cross-node RDMA typically incurs higher latency due to additional network hops and cable propagation, our loopback configuration provides an idealized, best-case baseline for RDMA. Consequently, the performance gains reported for SAC represent a conservative estimate of its real-world advantages.

A.3 Software and Framework Configuration

SGLang Settings. We deploy SGLang with the following hyperparameters:

- Tensor Parallelism (TP): 8
- Data Parallelism (DP Attention): 8
- Memory Configuration: `bf16` KV cache precision, `mem-static-fraction` set to 0.85.

HiSparse Settings. We configure HiSparse with a *top-k* value of 2048 and a device buffer pool of 6144. As standard HiSparse currently does not support Radix Cache, we implemented a custom Radix Cache integration within HiSparse. In this configuration, the KV cache is entirely offloaded to the designated memory backend (CXL or RDMA).

B Extended Literature Review

B.1 Related Works: CXL-based KV Cache Management System

The limitations of RDMA-based disaggregated memory pools, specifically high access latency, complex communication protocols, and synchronization overhead, have recently spurred the development of CXL-based architectures for LLM serving. Two prominent systems, **Beluga** (Yang et al., 2026) and **TraCT** (Yoon et al., 2025), have established the foundation for using CXL as a high-performance substrate for KV cache.

B.1.1 Beluga: Scalable Shared Memory via CXL Switches

Beluga (Yang et al., 2026) is the first system to leverage CXL 2.0 switches to enable GPU clusters to access a shared, disaggregated memory pool. By replacing the network-based RDMA protocol with a memory-semantic interface, Beluga allows GPUs to perform direct peer-to-peer (P2P) transfers and fine-grained, non-contiguous access via custom CUDA kernels.

A key contribution of Beluga is its characterization of the "indirect host-staged data path" in RDMA, where data must move through a host "bounce buffer," incurring significant latency. Beluga eliminates these extra copies and simplifies the control path by unifying data transfer within the GPU's native CUDA stream. While Beluga acknowledges the efficiency of CXL for sparse KV cache access patterns, its primary focus remains the management of large-scale, contiguous KV cache blocks within a unified address space to simplify scheduling and capacity scaling.

B.1.2 TraCT: Rack-Scale Serving with Direct GPU-CXL DMA

TraCT (Yoon et al., 2025) focuses on disaggregated LLM serving, specifically separating the compute-intensive prefill phase from the latency-critical decode phase. It uses CXL shared memory as both a KV-transfer substrate and a rack-wide prefix-aware cache. Unlike previous systems that rely on RDMA to move KV tensors between prefill and decode workers, TraCT enables GPUs to write and read KV blocks directly via CXL load/store and DMA operations, entirely eliminating the NIC hop.

TraCT provides critical software solutions for operating on non-coherent CXL Type-3 devices through three key mechanisms. First, it implements two-tier synchronization to bound contention by utilizing local DRAM locks in conjunction with a global shared-memory lock array. To ensure data visibility, TraCT employs fine-grained `clflush` instructions to guarantee metadata consistency across nodes . Finally, it facilitates zero-copy transfers by pinning the CXL memory region, which allows the CUDA runtime to treat and access it directly as page-locked host memory.

B.1.3 Distinguishing SAC from Existing CXL Systems

While Beluga and TraCT establish CXL as a superior alternative to RDMA for bulk KV block transfers, they primarily optimize for Dense Attention models where KV data is treated as monolithic blocks to be published or consumed. Our work, SAC (Sparse Attention on CXL), introduces a fundamental shift:

1. **From Transfer to On-Demand Fetching**: Whereas TraCT and Beluga focus on moving KV blocks, SAC utilizes CXL's near-DRAM latency to fetch only the **top-k KV entries** in real-time during the attention calculation. This addresses the Transmission Bottleneck (P1) that remains even in fast bulk-transfer systems.
2. **Mandatory vs. Optional Substrate**: In dense models, CXL is a performance optimizer; for Sparse Attention, SAC demonstrates that CXL is a necessity. By residing the full KV cache in the CXL pool and fetching only active entries, SAC resolves the Local Memory Wasting (P2) inherent in the "full prefetching" strategy used by RDMA-based systems.
3. **Fine-Grained Integration**: Unlike the block-level management in TraCT, SAC integrates directly with the SGLang runtime to handle the dynamic, runtime-determined indices of sparse attention models, utilizing CXL's zero-protocol overhead for discrete, cache-line granularity loads.

In essence, while prior work focuses on CXL as a **faster storage tier**, SAC treats CXL as a **direct extension of the GPU's memory hierarchy** specifically tailored for the non-contiguous access patterns of modern sparse LLM architectures.

B.2 CXL Techniques and its Applications

Protocol Evolution and Standards. Compute Express Link (CXL) was established by the CXL Consortium in 2019 to overcome the memory wall by leveraging the PCIe physical layer. While CXL 1.1 primarily supported point-to-point host-to-device connections, the protocol has significantly matured with subsequent versions. CXL 2.0 introduced hardware-managed switching and memory pooling capabilities, while CXL 3.0/3.1 further expanded these into a fabric-based architecture. This evolution allows CXL to provide high-bandwidth, low-latency memory sharing with hardware-managed load/store semantics, enabling access at cache-line granularity which is uniquely suited for fine-grained, sparse memory lookups.

Applications in Database Systems. In the domain of database systems, CXL is increasingly used as a compelling alternative to RDMA for cross-node memory access. Tigon (Huang et al., 2025) introduces a distributed database architecture specifically optimized for CXL pods, achieving high performance through the protocol's native memory semantics. Similarly, Yang et al. (Yang et al., 2025) explore unlocking the potential of CXL for disaggregated memory in cloud-native databases. These systems benefit from CXL to simplify data paths and transaction management compared to traditional network-based approaches.

Adoption in Cloud Platforms. Cloud infrastructure providers have adopted CXL to mitigate memory stranding and enhance resource utilization. Pond (Li et al., 2023) demonstrated an early production-scale CXL-based memory pooling system. Memtunnel (Guan et al., 2025) shifted the research focus toward the broader challenge of rack-scale orchestration. Zhong et al. (Zhong et al., 2024b) focuses on managing memory tiers within virtualized environments. RCMP (Wang et al., 2024) shows that CXL-based memory disaggregation significantly reduces software overhead and improves CPU efficiency in cloud-scale deployments.

Application to LLM Serving. Beyond general cloud workloads, CXL is becoming a critical enabler for Large Language Model (LLM) serving and Deep Learning (DL) acceleration. While Beluga (Yang et al., 2026) and TraCT (Yoon et al., 2025) primarily focus on prefix KV cache management for dense models, other works explore broader architectural optimizations. For instance, CXL-DL (An et al., 2024) investigates using CXL-attached memory to expand capacity for massive embedding tables in recommendation systems, which share the characteristic of sparse memory access with certain LLM configurations. Additionally, hardware-software co-designs like CXL-PNM (Kim et al., 2023) explore processing-near-memory within CXL Type-3 devices to accelerate LLM inference by reducing data movement.

B.3 Hardware Cost Analysis: CXL vs. RDMA

To quantitatively evaluate the cost-effectiveness of the SAC architecture, we provide a comparison between RDMA-based networking and CXL-based memory pooling, based on commercial hardware specifications. As shown in Table 2, CXL-based components significantly reduce the cost of host interface cards and switches at the device level.

Table 2: Hardware Cost Analysis (Yang et al., 2026)

Component	**RDMA-based (RoCE)**	**CXL-based**
Interface on Host	ConnectX-7	PCIe/CXL Adapter
PCIe Lanes	x16	x16
Price	$1,745	$210
Switch	Mellanox RoCE Switch	XConn CXL Switch
Ports/Capacity	40 x 200Gbps	—
Price	$16,000	$5,800
$/(64GB/s)	**$800**	**$218.75**

Beyond direct component pricing, SAC offers significant system-level cost benefits:

- **Memory Density and Type**: Unlike traditional DIMM memory in the host, CXL memory devices can use lower-density chips, reducing the cost per GB.
- **Resource Disaggregation**: CXL memory pools separate memory from CPU resources, allowing multiple servers to share memory and improve utilization in cloud environments.
- **Local Memory Reduction**: By keeping the full KV cache in the CXL pool and fetching only the required top-k entries on demand, SAC eliminates the need for over-provisioning expensive TB-level local DRAM on every compute node, further lowering the total cost of ownership (TCO) for sparse attention model serving.

C SGLang HiSparse: The Implementation Basis of SAC

The performance of SAC (Sparse Attention on CXL) is fundamentally built upon the architectural principles of **HiSparse** (Xie et al., 2026), a hierarchical memory management mechanism developed for the SGLang (Zheng et al., 2024) framework. This section details the motivation and implementation of HiSparse.

C.1 Motivation: Breaking the Memory Capacity Bottleneck

While sparse attention models reduce computational complexity by attending to a subset of tokens, they traditionally do not solve the local memory capacity limitation problem. In standard serving stacks, the KV cache for the full context must remain resident in GPU HBM to ensure low-latency access, even though only a tiny fraction of entries are active during any given decoding step.

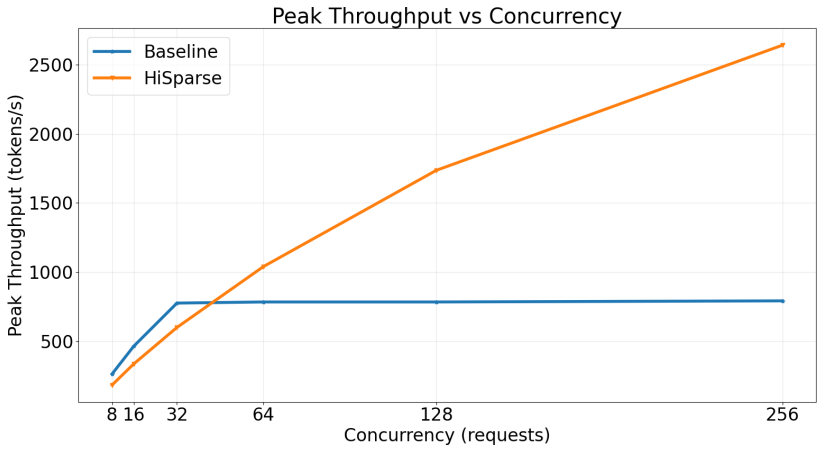

Figure 15: Benchmark results for the GLM-5.1-FP8 model using 32k-input, 8k-output queries on a PD-colocated 8×H200 deployment (Xie et al., 2026).

As illustrated in Figure 15, standard sparse attention implementations hit a memory capacity wall early, causing throughput to plateau as concurrency increases. HiSparse addresses this by introducing a hierarchical memory approach that offloads inactive KV cache entries to a secondary memory tier. This allows for near-linear throughput scaling; at 256 concurrent requests, HiSparse achieves over 3× the throughput of the baseline by decoupling token generation from local HBM capacity.

C.2 HiSparse Architecture and Workflow

The architecture of HiSparse revolves around a proactive offloading and reactive swap-in mechanism. As shown in Figure 16, the system maintains a subset of the KV cache in the local device buffer to minimize data movement on the critical path.

Central to HiSparse is a specialized Swap-in CUDA kernel designed to manage the data flow between the local HBM and the remote memory pool. The kernel performs three critical operations during each decoding step:

1. **Miss Identification**: It identifies which of the dynamically determined Top-k KV entries are missing from the local device buffer.
2. **LRU Eviction**: It selects eviction candidates from the hot device buffer using a Least Recently Used (LRU) policy to make space for incoming data.
3. **Page Table Update and Fetch**: It updates the internal page table mapping and triggers the fetch of the required KV entries from the remote pool to the local HBM.

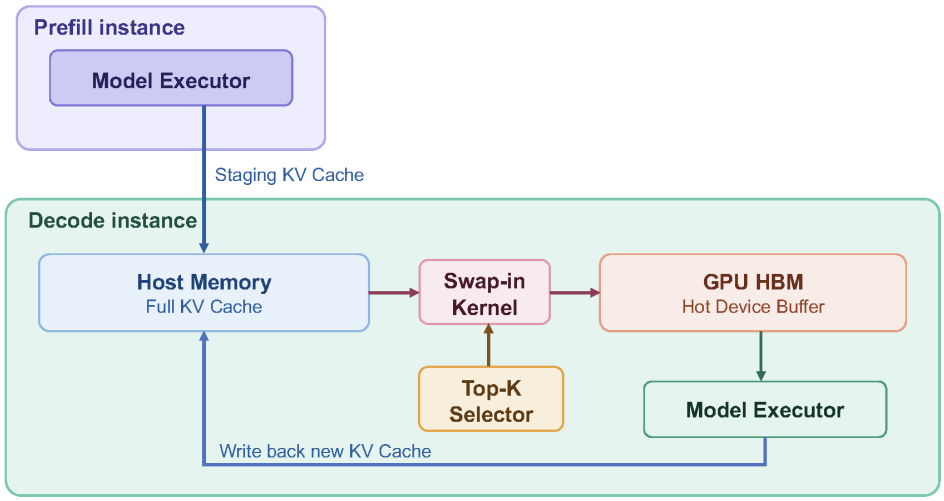

Figure 16: HiSparse workflow (Xie et al., 2026).

C.3 SAC Implementation on HiSparse

SAC orchestrates a hardware-software co-design by coupling the HiSparse framework with a CXL-centric memory hierarchy. This integration enables sparse decoding to leverage a fabric-attached, high-capacity KV tier, effectively decoupling the logical context length from the physical constraints of local memory.

The implementation is characterized by the following architectural components:

- **Hierarchical Memory Abstraction**: SAC redefines the HiSparse KV management stack by establishing CXL-backed disaggregated pools as the capacity tier. This architecture allows the authoritative KV corpus to scale independently of GPU HBM, providing the necessary volume for long-context windows while maintaining a logically unified memory fabric.
- **Unified Access Semantics**: By leveraging the memory-semantic nature of the CXL physical layer, SAC optimizes the CUDA kernels in HiSparse to treat disaggregated memory as a direct extension of its local hierarchy. This bypasses traditional staging overheads and allows for fine-grained, zero-copy interactions between the GPU and the remote KV buffer.
- **Sparsity-Aware KV Fetching**: SAC aligns the data movement strategy with HiSparse. The system performs reactive, sub-block fetching driven by layer-wise top-k selection, which exploits CXL's low-latency characteristics to satisfy real-time inference requirements with minimal I/O amplification.

D More Experimental Results and Analysis

D.1 Statistic Significance Evaluation

For the main evaluation in Section 5.1, we repeat each experiment 3 times and report the average of the results. Because the inference process is largely deterministic and conducted in a controlled environment, the experimental results exhibit minimal divergence. To quantify this stability, we calculate the Coefficient of Variation (CV), defined as $CV = (\sigma/\mu) \times 100\%$, where σ is the standard deviation and μ is the mean. Across all test cases (including various context lengths and concurrency levels), the CV for throughput remains below 2.1%, and the CV for latency metrics (TTFT and TBT) is consistently within 3.5%. The small standard deviation relative to the performance gains demonstrates that the observed advantages of SAC are statistically significant and not the result of transient hardware fluctuations or measurement noise.

D.2 Performance under Varying Output Lengths

In the main evaluation in Section 5.1, we fixed the output length at 1K tokens. This configuration represents a common production serving scenario where the generated output is significantly shorter than the input context. As AI agent workloads become increasingly mainstream, this disparity grows further, with input-to-output ratios often reaching 100:1 or even 200:1 (Qin et al., 2025). In these "long-input, short-output" scenarios, the inefficiencies of RDMA-based disaggregated systems—the transmission bottleneck and local memory wastage—become even more pronounced, as the entire KV cache must be fetched locally only to generate a relatively small number of tokens.

To evaluate the robustness of SAC across diverse workloads, we measure performance across various output lengths including 2K, 4K, and 8K tokens. The results are illustrated in Figures 17 through 22.

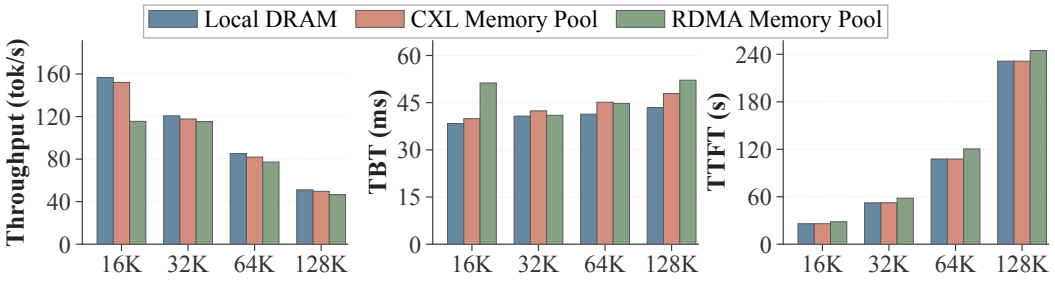

Figure 17: Performance comparison with 2K output length: Round-1 (Prefill).

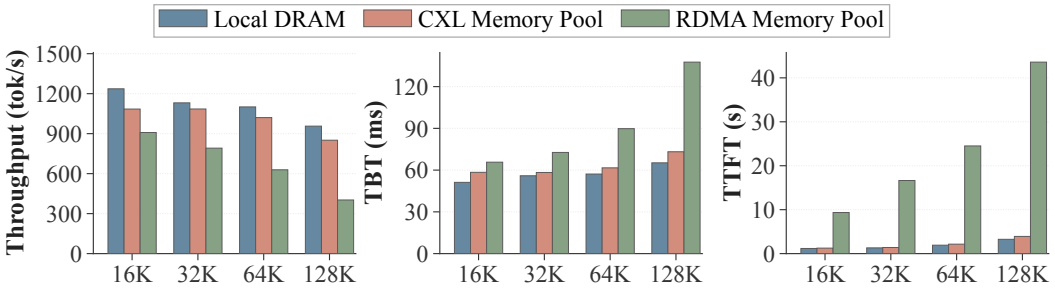

Figure 18: Performance comparison with 2K output length: Round-2 (Decoding).

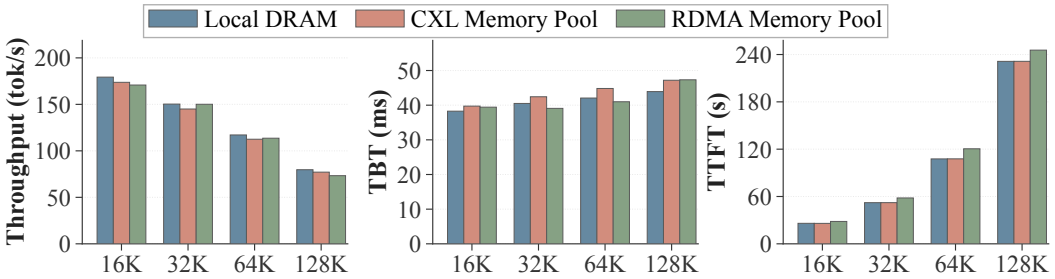

Figure 19: Performance comparison with 4K output length: Round-1 (Prefill).

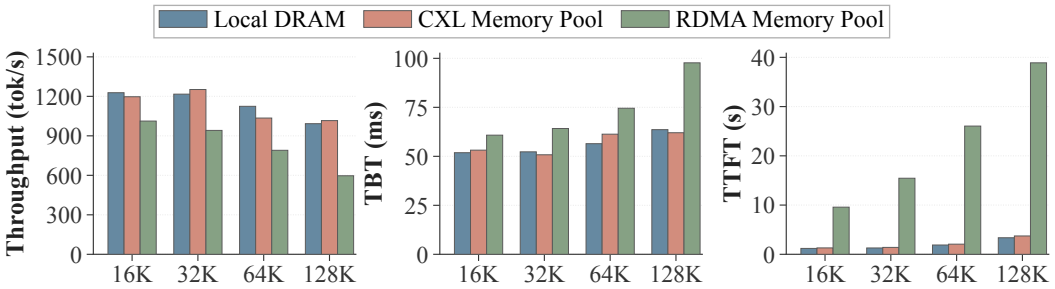

Figure 20: Performance comparison with 4K output length: Round-2 (Decoding).

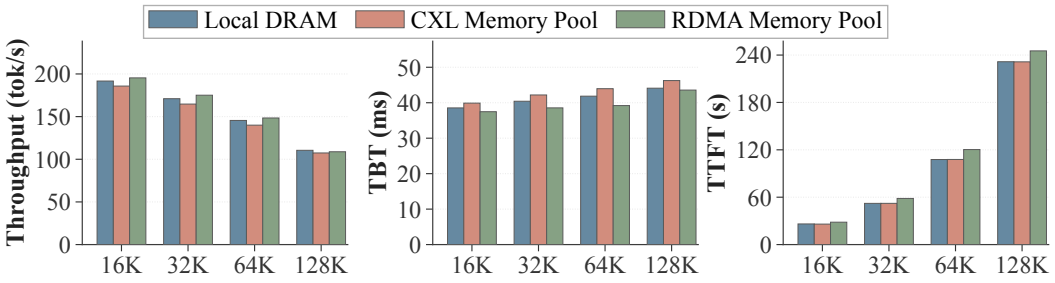

Figure 21: Performance comparison with 8K output length: Round-1 (Prefill).

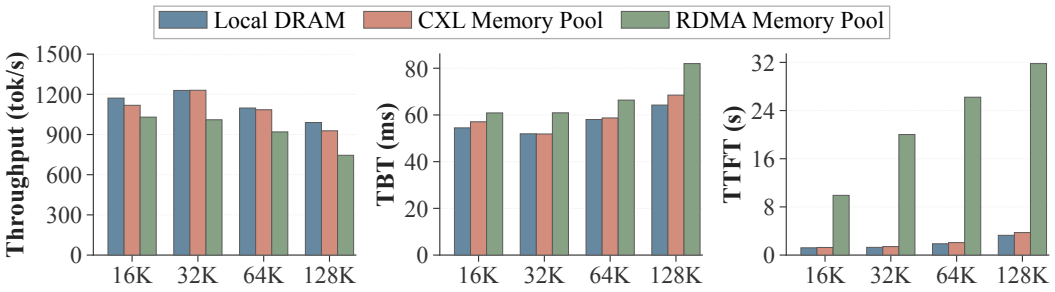

Figure 22: Performance comparison with 8K output length: Round-2 (Decoding).

The experimental results consistently align with the findings in Section 5.1. In Round-1 (prefill stage), all backends exhibit nearly identical performance, as the stage remains primarily compute-bound on the GPU. In Round-2 (decoding stage), SAC maintains a clear throughput advantage over RDMA across all tested output lengths, including the 8K configuration.

Notably, we observe that the throughput advantage of SAC is most significant at shorter output lengths. This is because for shorter generations, the massive fixed overhead of fetching the entire prefix KV cache via RDMA dominates the total request latency. As output length increases, this initial "transmission tax" is amortized over a larger number of generated tokens; however, SAC continues to outperform RDMA due to its superior handling of PCIe bus contention and lower TBT, as discussed in the main evaluation.

D.3 Tail Latency Analysis

While the main evaluation focuses on average throughput and mean latency, this section quantifies the tail behavior of SAC under increasing concurrency. Figures 23 and 24 illustrate the distribution of time between tokens (TBT) and time to first token (TTFT), respectively.

We observe that while the mean TBT grows modestly across all backends as concurrency increases, the p99 metrics for both TBT and TTFT rise at a significantly higher rate. This divergence indicates intensified request-level contention and latency variance under heavy loads. Notably, the CXL-based memory pool exhibits a wider gap between mean and p99 latencies compared to the local DRAM baseline at equivalent concurrency levels. This behavior is consistent with memory-side contention and arbitration overhead within the CXL fabric, which scales non-linearly with increasing request concurrency.

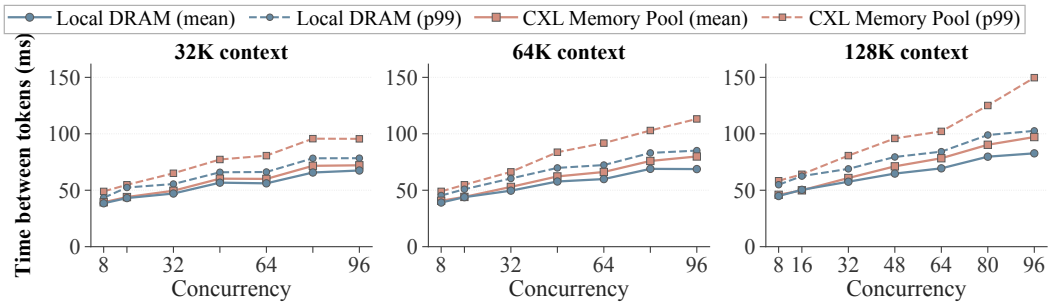

Figure 23: TBT Tail Latency: Comparison of Mean and p99 TBT across increasing concurrency.

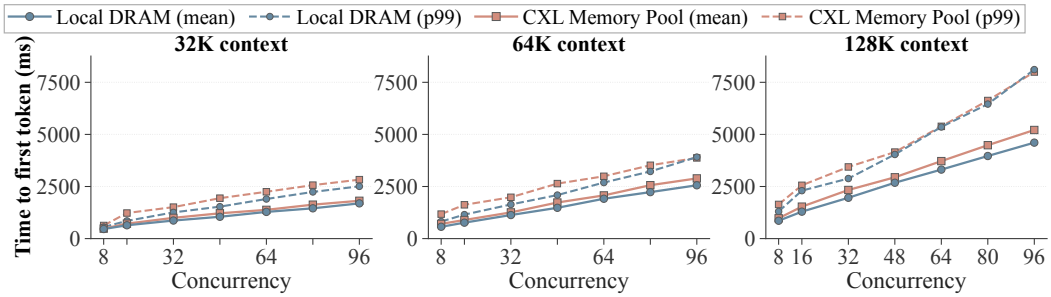

Figure 24: TTFT Tail Latency: Comparison of Mean and p99 TTFT across increasing concurrency.

D.4 Request-level Throughput

While the primary evaluation focuses on token-level throughput (tokens/s), we provide the corresponding request throughput (requests/s) here to offer a more comprehensive performance profile. As illustrated in Figure 25, SAC achieves request throughput that closely approaches the local DRAM baseline across all evaluated output lengths. Furthermore, SAC significantly outperforms the RDMA-based disaggregated pool. This consistency between token-level and request-level metrics confirms the efficiency gains provided by SAC.

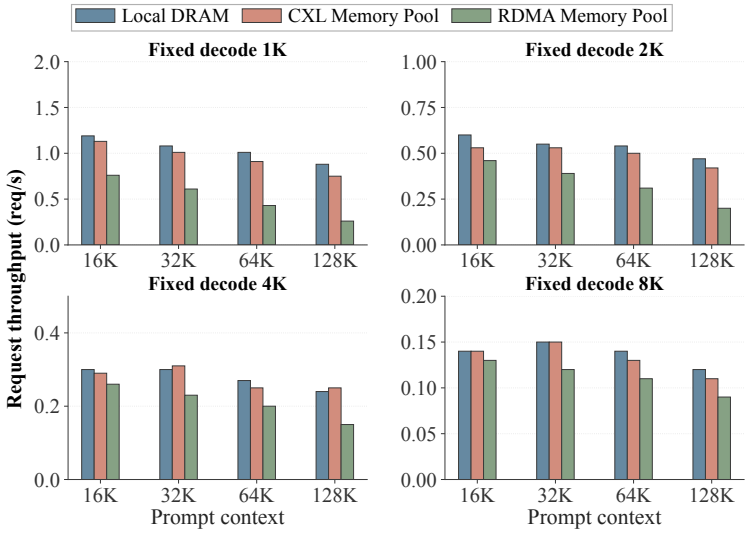

Figure 25: Request throughput (req/s) comparison across different backends and output lengths.

E Application Study: CXL Memory Pools for RL Rollouts

The main-text evaluation (Section 5) uses a single compute node attached to disaggregated CXL memory. Its two-round design separates initial KV construction from subsequent reuse, exposing the concurrent-throughput limit imposed by full-KV transfer in the RDMA path and the benefit of sparse access through CXL. This establishes the local mechanism that makes SAC useful for high-throughput decoding, but does not exercise a shared pool across a cluster of prefill (P) and decode (D) nodes.

A rack-scale shared L2 for inference. Compared with a host-DRAM L2, the CXL pool provides a rack-scale shared L2 that is disaggregated from compute nodes while retaining similar GPU read/write granularity and access latency (Yang et al., 2026). Multiple P and D nodes access the same resident KV through the rack's memory fabric. SAC's mapped-memory GPU kernels support fine-grained reads and writes to this tier, enabling sparse retrieval and incremental publication without first receiving an entire context. An RDMA-based distributed memory pool instead places KV across network-attached host memories and accesses it through explicit RDMA operations. This architectural distinction motivates the study; the evaluated RDMA baseline specifically uses direct full-KV P-to-D handoffs, without a separate pool server.

CXL's broader opportunity arises in multi-node inference clusters where high decode-throughput demand makes repeated KV handoffs a bandwidth bottleneck. When many long-context requests arrive at D, receiving a complete KV copy for each request can delay decoding even though sparse attention uses only a small subset of that history at each step. A centralized, disaggregated CXL memory pool with load/store semantics lets P and D access the same resident KV: P publishes new KV incrementally, and D fetches only the entries required by top-k attention that are absent from its local buffer. The key question is therefore *when can this reduction in D-side transfer demand improve throughput and completion time across a multi-node inference cluster?*

We select the generation phase of RL rollouts as a representative application because it prioritizes aggregate throughput across many concurrent trajectories (Sheng et al., 2024). These trajectories often share an initial prompt and repeatedly extend long histories, creating frequent P-to-D KV handoffs. As P supplies more calls to D, transferring their full histories can constrain decode throughput. This makes rollouts a natural setting for CXL's fine-grained sparse access and the high-throughput decoding supported by HiSparse (Xie et al., 2026). Because CXL devices are scarce in our test environment and a physical multi-node fabric requires substantial integration, we extend hardware-based component models to a trace-driven cluster simulation. The experiments increase P provisioning from 1P1D to 2P1D and 4P1D and compare equal-bandwidth controls as well as explicit RDMA NIC provisioning against one CXL interface per node.

E.1 RL Rollouts: Opportunities and Workload

Repeated handoffs and shared histories. In agentic rollouts, a multi-turn trajectory alternates between LLM generation and new observations from tools or the environment, as illustrated by the released SWE-agent trajectories (Nebius, 2024). Each subsequent LLM call extends the conversation, reuses its existing KV, and requires another P-to-D handoff. Our direct-RDMA baseline sends the full input KV from P to D on every call. With SAC, historical KV remains in the shared CXL pool: P publishes only newly computed input KV, and D reads sparse misses during decoding. This removes repeated full-prefix reception at D, while retaining the costs of P-side history reads, incremental publication, sparse reads, and new-token writes.

Trajectories derived from the same prompt can also share immutable prefix KV. CXL keeps one canonical prefix accessible to all P and D nodes. The RDMA baseline deduplicates prefixes within each home P node, but materializes a complete temporary KV copy for each D-side call. CXL can therefore reduce both repeated handoff traffic and complete D-local replicas. Their effect on rollout completion time depends on computation, transfer contention, and queueing.

Nebius-based workload and length sensitivity. We derive the multi-turn call graph and continuation-length estimates from our archived processing of Nebius SWE-agent trajectories (Nebius, 2024), with a constructed 128,000-token initial prompt for each prompt group. Call order, dependencies, and the archive's pre-expansion environment increments are preserved. Lengths retain the archive's

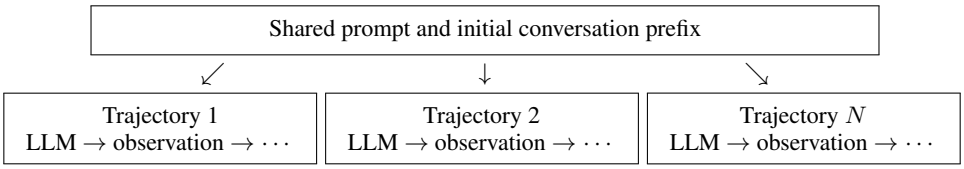

Figure 26: Rollout trajectories share an initial prefix and retain growing private histories across successive LLM calls.

character-based calibration; target-model tokenization and shared-prefix identity are not independently revalidated.

We select 32 prompt groups with 16 trajectories each, giving 512 trajectories and 20,175 LLM calls. The main experiment retains the archived output lengths: 2,454,688 tokens in total, averaging 121.7 tokens per call. Subsequent environment increments average 428 tokens. The 128K initial context is synthetic; this is a Nebius-based long-context workload, not a recorded 128K RL deployment.

For output-length sensitivity, we scale each call's output to 50% or 25% of its archived length, round to the nearest integer with a minimum of one token, and recompute all subsequent context and KV lengths. The resulting mean outputs are 61.1 and 30.6 tokens. This preserves relative length variation and a long tail without imposing a fixed output cap: the longest outputs remain 2,424 and 1,212 tokens, respectively. Call dependencies and environment increments stay fixed; generated text and downstream environment contents are not regenerated. These length-only sensitivities vary generation work per handoff.

Paired backends receive identical inputs. Every rollout starts cold at time zero: initial prompt KV must be computed and published through the modeled paths, and all initialization remains included in the reported makespan. Increasing P provisioning from 1P1D to 2P1D and 4P1D tests whether a single D receiver becomes constrained as KV supply increases; a large submitted batch alone does not establish receiver saturation.

E.2 Resource and Cost Model

From hardware profiles to cluster simulation. We build the node-level model from hardware profiles of the SGLang/HiSparse execution path (Zheng et al., 2024; Xie et al., 2026). We profile prefill and decode computation on eight H200 GPUs, and separately measure KV publication, sparse reads, and new-token writeback on real DevDAX-backed CXL hardware. These are additional component profiles, distinct from the main-text H20 end-to-end evaluation. The resulting compute curves and KV-path/access evidence define the node-level components that we replicate across P and D nodes. Table 3 distinguishes these measurements from the extrapolation used in this study.

Table 3: Hardware evidence and its role in the cluster model. Component profiles come from separate machines; cluster results are simulated.

Component	Hardware evidence	Use in this study
Prefill / decode	H200, TP8 and attention DP8; context-length sweeps	Interpolated compute curves; explicit batch and length extrapolation
KV movement / access	CXL DevDAX component measurements; HiSparse access and swap-in profiles	Phase boundaries and KV-access model; payloads derived from the modeled workload
Cluster transport	Per-node interface specifications and endpoint paths	Payload / bandwidth with shared-resource contention; no measured multi-host timing claim

Nodes and compute. Each P or D node contains eight GPUs, using TP8 and attention DP8/TP1, and P batches up to eight calls. The isolated H200 profiles exclude host offload, so separately scheduled transfers do not duplicate the measured compute time. Decode step time is $F_D(8\bar{L})[1 + \alpha(b-1)]$, where F_D is the measured eight-request context-length curve, $\bar{L}$ is the active requests' mean context, and b is the largest actual per-GPU occupancy. We use $\alpha = 0.2$ as a shared batch-extrapolation

assumption for CXL and RDMA. We interpolate within the measured range and use linear decode and quadratic prefill tails outside it; cached prefill scales with the uncached-token fraction. Transfers are scheduled separately. Prefill produces the first output token, leaving $O - 1$ decode steps for an output of length O.

KV placement and movement. The component model uses 89,856 KV bytes per token ($78 \times 576 \times 2$). Under CXL, P reads required historical KV from the pool, computes the new input suffix, and publishes only that suffix. D reads missing sparse KV and incrementally writes newly materialized output KV to the pool. Under direct RDMA, the trajectory's home P retains its history, restores it locally for prefill, stages the input KV, and transfers the complete input KV directly to D. D serves sparse misses from its temporary host-DRAM copy. At call completion, newly materialized output KV is returned to P and the D-local copy is released; no complete session KV is retained at D across calls. These paths use the same HiSparse policy and computation function.

Interfaces and contention. CXL uses one modeled PCIe 5.0 x16 interface per P or D node, with a 64 GB/s payload ceiling per direction. RDMA uses 2, 4, or 8 single-port 100 Gb/s NICs per node, corresponding to aggregate line rates of 25, 50, or 100 GB/s per direction. This rate corresponds to documented single-port 100 Gb/s adapters (NVIDIA, n.d.); we use the line rate as an optimistic payload ceiling, without an efficiency penalty. These are provisioning comparisons, not equal-bandwidth comparisons: in particular, eight NICs provide more aggregate bandwidth than the CXL interface. A separate 1×64 GB/s control gives both backends identical interface resources.

Each NIC is an independent resource with fixed affinity to an equal subset of the eight GPU ranks. Transfers on the same interface and in the same direction share bandwidth; opposite directions are independent. CXL accesses occupy the requesting node's interface, while RDMA handoffs occupy both source transmit and destination receive interfaces. The simulator computes transfer time from logical payload and equal-weight max-min bandwidth sharing, with 10 μs RDMA setup and ready events. The CXL hardware profiles establish component behavior; their original device-specific latency floors are not superimposed on these provisioned-bandwidth runs. Pool-side and fabric resources are assumed sufficient, without a global bandwidth cap. Local GPU-link, switch, and NUMA resources retain the same component-model limits for both backends.

Capacity and sparse accesses. P's temporary buffer remains bounded. D-side host DRAM is assumed sufficiently provisioned: complete temporary KV replicas are allocated, tracked, and released, but their size does not impose an admission cap. The modeled sparse-KV working set in GPU HBM has a fixed budget independent of full-context length; this is not a budget for total model HBM usage. Every rank has 16 fixed working-set slots of 4,096 tokens each, with top-$k = 2,048$ per active request. All concurrency settings and both backends receive this same sparse-KV budget; a call retains one slot throughout decoding. Persistent history capacity covers the whole rollout's final KV bound, with the same aggregate budget in the CXL pool and the distributed P-side RDMA histories.

A frozen component-access capture drives a deterministic block proxy that preserves cold-start and evolving miss behavior. Each slot has a bounded LRU working set, and shared-prefix and private-trajectory blocks have separate identities. The proxy uses 64-token blocks, correlated layer accesses, and extrapolation beyond measured steps; it is not an exact replay of the selected workload's layer-wise top-k indices. Reported traffic is logical KV payload rather than measured physical bus traffic.

E.3 Simulation Workflow and Accounting

Admission, prefill, and handoff. All 512 trajectories enter their home P queues at time zero, without a session-concurrency watermark. An idle P node selects ready calls in first-come, first-served order, subject to its temporary buffer, and assigns each call to a D rank with the fewest outstanding calls. It restores history and computes the uncached suffix. The P node can start a new prefill after computation finishes while the preceding handoff proceeds; its buffer remains reserved until that handoff completes. Compute/publication overlap is explicit, and shared transfer resources still contend. Prefixes become reusable after publication; cold requests already admitted to P are not coalesced into a single shared prefill computation.

Continuous decoding. A call enters its assigned D node's ready queue after handoff. At each forward boundary, available per-rank slots admit ready requests. Every active request advances by

one token; completed requests release their execution slots, and subsequent calls reenter their home P queues after the required KV state is available. In the RDMA path this includes the delta return to P. Forward synchronization is local to a D node, rather than a barrier over the entire rollout. The simulator records both configured concurrency limits and actual per-rank occupancy, so additional queued requests are not counted as concurrently executing requests.

Completion and auditing. The rollout ends only after all trajectories complete, events and transfers drain, and KV allocations are released. External tool/environment service time is zero; reward evaluation, training, and weight updates are excluded. We report makespan, useful output-token throughput, component traffic, and simultaneous KV occupancy:

$$M_{\text{peak}} = \max_t \left[M_{\text{CXL}}(t) + M_{\text{P,history}}(t) + M_{\text{P,buffer}}(t) + M_{\text{D,DRAM}}(t) \right]. \tag{1}$$

This metric excludes weights, the separately budgeted common HBM working sets, metadata, and allocator overhead. It is not the sum of separate component peaks or a procurement-cost estimate. Audits verify completion sets, output-token counts, incremental CXL publication, full RDMA handoffs, delta returns, HBM and P-buffer bounds, and allocation–release balance. We separately accumulate prefill compute for initial, continuation, and mixed batches; these node-time sums can overlap and are not interpreted as additive wall-clock phases.

E.4 Experimental Configuration and Results

We simulate GLM-5.2 with FP8 weights and BF16 KV storage (Z.ai, 2026a). The KV payload model uses its 78 layers and 576 stored values per token per layer (512 latent KV values plus 64 rotary-position values).

Table 4: Rollout simulation configuration. Compute and cache budgets are matched between backends; interfaces are explicitly provisioned, while D host DRAM and pool-side bandwidth are provisioned assumptions.

Parameter	Value
P/D topologies / GPUs per node	1P1D, 2P1D, 4P1D / 8
Rollout trajectories / seed	512 / 42
Initial prompt / arrival	128K tokens / all trajectories at $t = 0$
Active decode limit per GPU	8; 4 and 16 at 50% outputs
Output-length scale	Unscaled / 50% / 25%
HiSparse top-k / tokens per slot	2,048 / 4,096
HBM KV budget per GPU, all settings	16 slots; 5.48 GiB
P temporary buffer per GPU rank	64 GiB
D host-DRAM admission cap	None; peak usage recorded
Persistent KV budget	Matched final-context bound within each pair
CXL interface per node	1×64 GB/s per direction
RDMA NICs per node	2 / 4 / 8 $\times$ 100 Gb/s (25 / 50 / 100 GB/s)
Equal-bandwidth control	Both backends: 1×64 GB/s
Pool/fabric aggregate bandwidth cap	None
Local GPU link / switch / NUMA	64 / 40 / 80 GB/s
RDMA setup / ready event	10 / 10 μs

Protocol. The main experiment retains unscaled outputs at 1P1D, 2P1D, and 4P1D, and sweeps 2/4/8 RDMA NICs per node against one CXL interface. We then repeat the NIC sweep at 4P1D with 50% and 25% output lengths, giving 15 RDMA configurations and five corresponding CXL controls. The active decode limit is eight requests per GPU. All comparisons use identical inputs, compute/cache policies, and node counts within each workload/topology setting; only the stated remote-interface provisioning differs. We also retain the equal-64-GB/s controls and their decode-limit sensitivity (4/8/16). All trajectories finish, including cold initialization and the drain tail. Results are trace-driven sensitivity simulations, not multi-host hardware measurements.

Main experiment: unscaled outputs. Figure 27(a) retains the archived generation lengths and increases P supply to one D node. Every RDMA configuration uses the stated NIC count at *each* P and D node; CXL uses one interface at each node. Speedup is T_R/T_C, so values below one mean CXL is slower. Node counts are matched within each group; increasing P across groups changes total GPU provisioning.

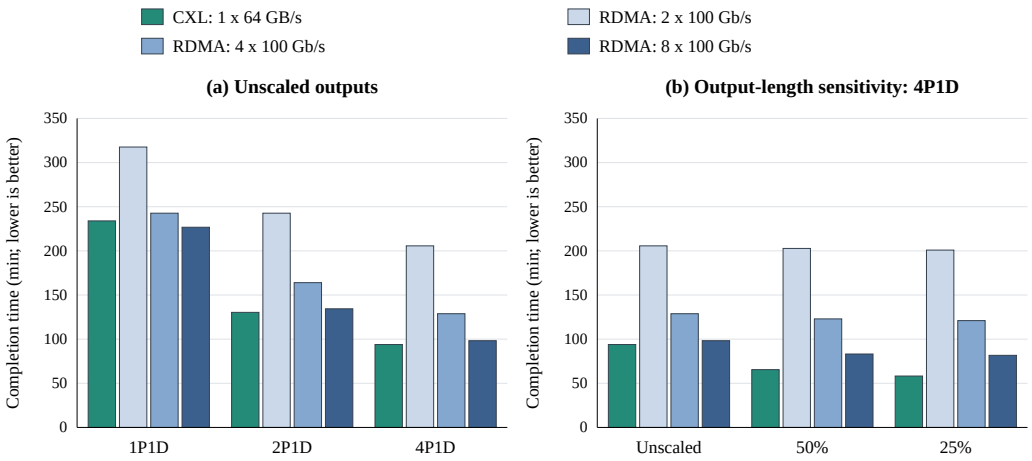

Figure 27: Simulated cold-start rollout completion time (lower is better), with a limit of eight active requests/GPU. (a) Unscaled outputs with increasing P supply. (b) Output-length sensitivity at 4P1D. Each legend entry specifies interfaces at every P and D node; each RDMA NIC supplies 100 Gb/s. Both panels include initialization and the completion tail and use the same zero-based vertical scale.

More prefill supply increases the benefit of sparse retrieval. Figure 27(a) shows that CXL's advantage grows as more P nodes supply a single D node. Against two RDMA NICs per node, CXL achieves 1.358×, 1.862×, and 2.190× speedup at 1P1D, 2P1D, and 4P1D, respectively. With four NICs, the corresponding speedups are 1.037×, 1.258×, and 1.372×. At 4P1D, CXL completes the rollout in 93.92 minutes, compared with 205.68 and 128.83 minutes for the two- and four-NIC RDMA configurations. Increasing P supply raises the demand for KV delivery at D, making the cost of receiving complete histories more consequential. CXL reduces this demand by fetching sparse entries from the shared pool.

Additional RDMA bandwidth narrows the gap. With eight NICs per node, CXL is 3.17% slower at 1P1D, but achieves 1.031× and 1.046× speedup at 2P1D and 4P1D. Thus, at unscaled output lengths, the benefit is strongest when full-KV handoffs are constrained by receiver bandwidth; it is modest when the RDMA path has sufficient provisioning relative to P supply.

Shorter generations increase the cost of handoffs relative to decoding. Figure 27(b) fixes 4P1D and scales outputs to 50% and 25%, preserving the call graph and rebuilding subsequent contexts. CXL completion time falls from 93.92 minutes to 65.48 and 58.23 minutes. With two RDMA NICs per node, completion time remains near 200 minutes (205.68, 202.77, and 200.88 minutes), so CXL's speedup increases from 2.190× to 3.097× and 3.450×. With four NICs, RDMA takes 128.83, 122.95, and 121.01 minutes, giving CXL speedups of 1.372×, 1.878×, and 2.078×, respectively. These correspond to 27.10%, 46.74%, and 51.89% reductions in completion time.

Even with eight RDMA NICs per node (100 GB/s), CXL achieves 1.046×, 1.270×, and 1.403× speedup at unscaled, 50%, and 25% output lengths using one 64 GB/s interface. Shortening generation reduces computation per call while leaving a large input history to transfer at each handoff. Full-KV delivery therefore accounts for a greater share of rollout time, whereas CXL fetches only the sparse entries needed during decoding. Together, the two panels show that a shared CXL pool is particularly useful when sufficient P supply and frequent long-context handoffs make D-side KV reception a limit on aggregate decode throughput.

Bottleneck changes. Table 5 uses the middle provisioning point (four NICs) to relate performance to D-side pressure. Receive utilization is payload divided by receive capacity and makespan, averaged across interfaces. Contention is the mean fraction of time that an interface serves overlapping transfers. Decode occupancy is the time-average executing-request count per GPU, including idle intervals.

Table 5: C/R denotes CXL/RDMA. RDMA has four 100 Gb/s NICs/node; contention and handoff columns describe RDMA.

Scale / P/D	D RX (C/R)	Contended	Handoff (s)	Decode avg. (C/R)
Unscaled / 1P1D	1.5/35.1%	34.6%	5.60	0.91/0.93
Unscaled / 2P1D	2.7/52.0%	49.4%	11.70	2.42/1.52
Unscaled / 4P1D	3.7/66.2%	63.0%	23.46	3.40/2.02
50% / 4P1D	3.5/68.5%	66.1%	22.71	2.44/0.80
25% / 4P1D	2.8/69.1%	67.0%	25.12	1.03/0.36

Repeated full receptions delay KV availability and can leave D GPUs underfilled even when requests are queued. The relevant pressure is therefore on the receiving path, rather than evidence of continuously saturated GPU computation. CXL avoids most of this D-side demand by serving sparse misses from resident shared KV.

Equal-bandwidth and decode-concurrency controls. The equal-64-GB/s controls use one identical interface per node for both backends. Table 6 retains these controls, separating the shared-pool mechanism from the unequal aggregate rates in the NIC sweep.

Table 6: Identical one-interface, 64 GB/s-per-direction controls; limit eight/GPU. These rows are not RDMA NIC-line-rate configurations.

Scale / P/D	CXL (min)	RDMA (min)	Speedup	Time reduction
Unscaled / 1P1D	233.96	225.56	0.964×	-3.73%
Unscaled / 2P1D	130.36	147.03	1.128×	11.34%
Unscaled / 4P1D	93.92	112.73	1.200×	16.68%
50% / 4P1D	65.48	101.80	1.555×	35.68%
25% / 4P1D	58.23	100.79	1.731×	42.23%

At equal bandwidth, the unscaled 1P1D case remains 3.73% slower with CXL, while 4P1D improves by 16.68%; the benefit is not universal. At 4P1D and 50% outputs, increasing the decode limit from 4 to 8 to 16 gives CXL/RDMA speedups of 1.464/1.555/1.568×. Thus both backends support concurrent decoding, while handoff readiness determines how much of that capacity is useful. The one- and two-P controls at 50% outputs remain close to 0.97× and 1.19× across these limits.

E.5 Memory Savings and Cost Implications

CXL can improve rollout resource efficiency in two ways: fewer complete KV replicas reduce memory demand, and sparse access reduces the bandwidth needed at D. Memory pooling has been studied as a way to improve DRAM utilization and cost (Li et al., 2023); here we quantify the opportunity created by shared KV and avoided full handoffs. We price memory at the same unit rate for both backends and separately account for host interface cards and rack switches, using the same compute nodes within each pair.

Memory footprint and transfer demand. Figure 28 compares 4P1D with four 100 Gb/s RDMA NICs per node against one 64 GB/s CXL interface. Across unscaled, 50%, and 25% outputs, CXL uses about 1.32 TiB of peak simultaneous KV storage, compared with 6.20, 5.60, and 5.33 TiB for RDMA: a 75.2–78.6% reduction. Shared prefixes and persistent pool histories avoid complete D-side replicas. The metric includes the host staging buffers on both paths; the common HBM working sets remain unchanged. These are occupied logical bytes. Installed DRAM also depends on placement, allocation overhead, and per-node reserve capacity.

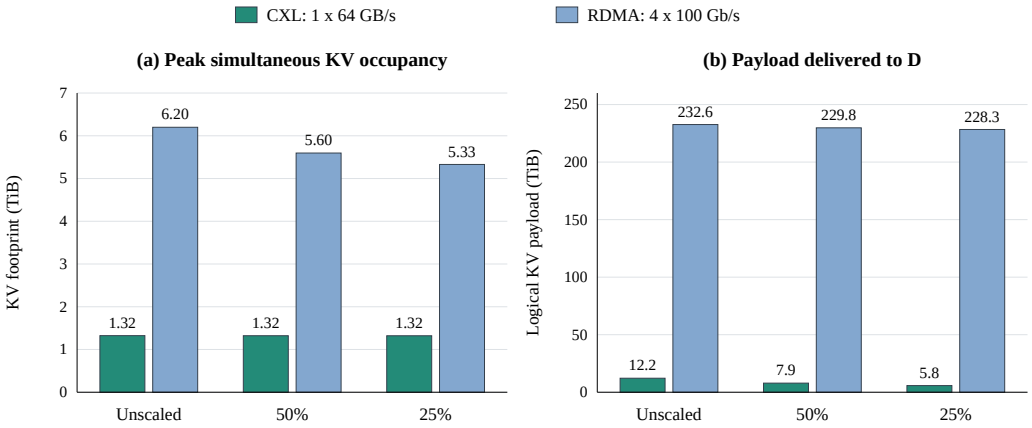

Figure 28: Resource demand at 4P1D with a limit of eight requests/GPU, including cold initialization. (a) Peak simultaneous KV occupancy across the cluster, excluding weights and common HBM working sets. (b) Cumulative input-KV handoff payload for RDMA and sparse-read payload for CXL at D. Both metrics are lower-is-better; the two panels have different units of accounting and vertical scales.

D-received payload falls by 94.8–97.5%, reducing the demand that drives receiver contention. CXL still incurs 226.6/223.7/222.3 TiB of P-side historical reads for the three output scales, as well as incremental writes. Thus the D-side reduction is not an equivalent reduction in total cluster traffic; the pool and its fabric must carry the remaining accesses.

Endpoint and bandwidth provisioning. At 4P1D, CXL uses five host-facing interfaces in total, while the 2/4/8-NIC RDMA configurations use 10/20/40 single-port adapters and active network ports. Relative to these configurations, CXL reduces the host-facing interface count by 50/75/87.5%. These counts do not equate the prices of a CXL interface and an RDMA adapter. The strongest provisioned RDMA point offers 100 GB/s per node, versus CXL's 64 GB/s, yet CXL is 1.046/1.270/1.403$\times$ faster at unscaled/50%/25% outputs. This demonstrates lower host-interface provisioning for equal or better completion time within the tested 4P1D settings, without assuming equal PCIe-lane budgets or proving a minimum-cost RDMA configuration.

Unit prices and component costs. Table 7 makes the cost comparison explicit for the unscaled 4P1D workload. We use the same DDR5 reference price on both paths: \$927 per 64 GiB, the finalized Q1 2026 64GB RDIMM benchmark reported in Counterpoint's April tracker (Counterpoint Research, 2026). We interpret the module capacity in binary units, giving \$14,832/TiB. For the single-port 100 Gb/s MCX651105A-EDAT, we use a verified seller listing of \$720 (Router-Switch.com, 2026), rather than applying the earlier appendix's \$1,745 dual-port 200 Gb/s NIC price to a different model. From Table 2 and its source (Yang et al., 2026), we retain \$210 per CXL host adapter, \$16,000 per RDMA rack switch, and \$5,800 per CXL rack switch. The latter is a reported B1 sample price. These are dated reference prices, not a contemporaneous production-system quotation.

Table 7: Illustrative component costs for unscaled 4P1D. Costs are USD; memory is priced identically at $927 per 64 GiB. Switch costs are literature budgets (CXL: B1 sample price). Subtotals include only the three priced components.

Item	CXL	RDMA: 2 NICs	4 NICs	8 NICs
Bandwidth/node (GB/s)	64	25	50	100
Host cards (count)	5	10	20	40
Host cards (USD)	1,050	7,200	14,400	28,800
One rack switch (USD)	5,800	16,000	16,000	16,000
Peak KV (TiB)	1.324	5.853	6.199	7.192
Memory at common price (USD)	19,639	86,817	91,942	106,677
Priced subtotal (USD)	26,489	110,017	122,342	151,477
Rollout completion (min)	93.92	205.68	128.83	98.26

We budget one complete rack switch for each backend: the cited RDMA switch provides 40 ports, sufficient for the 10/20/40 host links in this sweep. The subtotal is $C_{\text{priced}} = pM_{\text{peak}} + N_{\text{cards}}c_{\text{card}} + c_{\text{switch}}$, with the same p for CXL and RDMA. Memory costs value the observed logical peak, including temporary host buffers, without module rounding or placement headroom. Each RDMA column uses its own observed peak because interface provisioning changes queueing and replica lifetimes. Cables, optics, pool-memory controllers/enclosures, and operating costs are outside this component subtotal; common compute and HBM are excluded from both sides.

Against four NICs per node in the unscaled workload, the priced difference comprises $72.3K in KV memory, $13.35K in host cards, and $10.2K in switch budget. CXL's listed-component subtotal is 78.3% lower while the rollout is 1.372× faster. Across the three output scales at this NIC provisioning, the memory component alone saves $59.4–72.3K at the common unit price. This ties the cost advantage to reduced replication and interface demand, without assuming cheaper DRAM for CXL or equating the subtotal reduction with whole-system TCO.

E.6 Conclusions and Applicable Regimes

The simulations identify a concrete setting in which a centralized CXL memory pool benefits multi-node inference: high-throughput sparse decoding with long-context requests, where repeated P-to-D KV handoffs become a bandwidth bottleneck at the decode nodes. SAC keeps the full histories in a shared pool accessible through load/store semantics. P publishes newly computed KV, while D retrieves only the top-k entries missing from its local buffer. This replaces repeated full-context reception with fine-grained sparse reads, reducing D-side bandwidth demand and the temporary KV copies needed to sustain concurrent decoding.

RL rollouts exemplify this setting because many trajectories repeatedly alternate between prefill and generation while retaining growing KV histories. In the 4P1D study, CXL achieves 1.372× speedup with unscaled outputs and 1.878× and 2.078× with 50% and 25% outputs against four RDMA NICs per node. The larger gains as handoffs become more frequent relative to decode computation show where the shared pool is most useful: when KV delivery limits the throughput that D could otherwise serve. By fetching sparse top-k entries on demand, CXL relieves this bandwidth bottleneck and allows more of the available decode capacity to be used. Under the modeled resources and full-handoff baseline, these results support CXL pooling for inference clusters with high decode-throughput demand and frequent long-context KV handoffs; gains are smaller when prefill supply or computation dominates completion time.