CXL-MoE: Accelerating Sparse Model Inference through Intelligent Expert Offloading to CXL Memory

Abstract

The proliferation of Mixture-of-Experts (MoE) Large Language Models (LLMs) represents a significant leap in model capacity and computational efficiency. However, their massive parameter counts create a severe memory bottleneck, with model sizes often exceeding the High-Bandwidth Memory (HBM) available on a single accelerator. This paper introduces CXL-MoE, a novel inference system that addresses this memory wall by strategically offloading inactive expert parameters to Compute Express Link (CXL) memory expanders. CXL-MoE is architected to exploit the unique properties of the CXL interconnect, which provides a new tier of low-latency, high-capacity memory. The core of our contribution is a CXL-aware predictive prefetching algorithm that dynamically fetches required experts from CXL into a software-managed GPU cache. This algorithm adapts its prediction horizon to CXL's latency profile and is designed to leverage CXL's full-duplex communication channels, effectively hiding data transfer overhead. Through comprehensive evaluation using the Mixtral 8x7B model, we demonstrate that CXL-MoE significantly improves serving performance, increasing throughput by up to 1.8x and reducing per-token generation latency by 45% compared to state-of-the-art offloading systems that rely on the conventional PCIe interconnect to host DRAM.

---

1. Introduction

The Rise of Sparse Supermodels and the Memory Wall

The pursuit of greater capability in Large Language Models (LLMs) has led to an exponential growth in model size. This trend, however, is increasingly constrained by the practical limits of hardware, particularly GPU memory.[1] Mixture-of-Experts (MoE) architectures have emerged as a powerful paradigm to circumvent this scaling dilemma.[2] By replacing dense feed-forward network (FFN) layers with a larger set of sparsely activated "expert" sub-networks, MoE models can dramatically increase their total parameter count—and thus their capacity—while keeping the computational cost (FLOPs) per token constant.[3]

The Mixtral 8x7B model serves as a prominent example of this approach. It contains eight distinct 7-billion-parameter experts per MoE layer, resulting in a total of 47 billion parameters. During inference, a gating network dynamically selects only two of these experts to process each token.[4] This sparse activation means that only 13 billion parameters are active for any given forward pass, achieving the performance of a much larger dense model with the computational footprint of a smaller one. Despite this computational efficiency, the full set of model parameters must reside in accessible memory. For Mixtral 8x7B, this amounts to approximately 94 GB in half-precision (FP16), a figure that exceeds the 80 GB of High-Bandwidth Memory (HBM) available on a state-of-the-art NVIDIA H100 GPU. This disparity creates a formidable "memory wall," forcing practitioners to use multiple GPUs for a single model instance even when the computational load does not warrant it, thereby limiting achievable batch sizes, restricting context lengths, and drastically increasing deployment costs.[6]

Critique of Existing Offloading Paradigms

The most common solution to this memory pressure is to offload the inactive experts from the scarce GPU HBM to a larger, more economical memory pool, typically the host CPU's main DRAM.[7] Systems such as fMoE and eMoE have pioneered this approach, employing sophisticated prediction algorithms to anticipate which experts will be needed for upcoming tokens and prefetching them from CPU memory into GPU HBM over the PCIe bus.[7]

While conceptually sound, this paradigm is fundamentally constrained by the characteristics of the underlying interconnect. Data transfers between the CPU and GPU via PCIe involve significant software stack overhead and are subject to high latency and limited effective bandwidth. This forces existing systems to adopt complex, long-horizon prediction models that attempt to forecast expert usage far in advance to hide the transfer latency.[11] Such models are computationally expensive and prone to mispredictions, which lead to two undesirable outcomes: either the required expert is not present in HBM, causing a high-latency stall as it is fetched on-demand, or the GPU cache is polluted with unneeded experts, evicting others that may be needed soon.

The Compute Express Link (CXL) Opportunity

Compute Express Link (CXL) is a transformative open standard interconnect that fundamentally alters the server memory hierarchy.[13] While built upon the PCIe physical layer, CXL introduces a suite of new protocols, most notably

CXL.mem, which allows a host processor to access memory on an attached device using standard load/store instructions with hardware-managed cache coherency.[15] This creates a new, intermediate tier of memory that is more capacious and cost-effective than HBM but offers significantly lower latency than traditional storage or even PCIe-based transfers to host DRAM.[16] The latency of accessing CXL-attached memory is comparable to that of a remote NUMA node in a multi-socket server, typically in the range of 200-400ns.[17] This positions CXL memory as an ideal offloading target for data that is "warm"—accessed too frequently for SSDs but too large to fit entirely in HBM—a category that perfectly describes the parameters of MoE experts.

Our Contribution: CXL-MoE

This paper introduces CXL-MoE, a novel inference system that co-designs the management of MoE experts with the unique architectural properties of CXL memory. Instead of treating the offload target as a generic, high-latency device, CXL-MoE leverages CXL's specific features to overcome the limitations of traditional CPU-offloading. The fundamental argument of this work is that CXL is not merely a faster data pipe; it represents a paradigm shift in memory architecture that necessitates a new approach to data management. Existing offloading systems are architected around the constraints of a legacy interconnect paradigm. For instance, recent work on CXLAimPod has shown that CXL's full-duplex channels can yield a 71.6% throughput improvement for LLM text generation by intelligently scheduling and overlapping read and write operations.[19] MoE inference is inherently a mixed read-write workload—it involves reading expert weights while simultaneously writing new key-value (KV) pairs to the KV cache. A system designed specifically for CXL can schedule expert prefetches (reads) to overlap with other memory operations, exploiting the full-duplex channel in a way that is impossible with traditional CPU offloading over a half-duplex DDR bus.

Our key contributions are:

1. A novel MoE inference architecture that utilizes CXL memory as a high-capacity, low-latency tier for storing the full set of expert parameters.
2. A CXL-aware predictive prefetching algorithm that adapts its prediction horizon and scheduling to CXL's specific latency profile and full-duplex communication capabilities.
3. An integrated on-GPU caching policy that intelligently manages the flow of experts between CXL and HBM, working in concert with the prefetcher to maximize hit rates and hide CXL access latency.
4. A comprehensive empirical evaluation demonstrating that CXL-MoE significantly outperforms traditional CPU-offloading methods in terms of throughput and latency for serving large MoE models.

---

2. Background and Motivation

Anatomy of MoE Inference

The Transformer architecture forms the backbone of modern LLMs. In an MoE-based Transformer, the standard FFN sub-layer within each decoder block is replaced by an MoE layer.[2] This layer consists of two primary components: a set of

N expert networks (e.g., N=8 in Mixtral) and a lightweight gating network, or router.[3] For each input token, the router computes a probability distribution over the experts and selects the top-

k experts (e.g., k=2) with the highest scores to process the token. The final output of the MoE layer is a weighted sum of the outputs from the selected experts.

Autoregressive inference with these models proceeds in two distinct phases: prefill and decode.[21]

- **Prefill Phase:** The input prompt tokens are processed in a single, large parallel batch. This phase is typically compute-bound, as the GPU's computational units are saturated by the large matrix multiplications involved.
- **Decode Phase:** Output tokens are generated one at a time. In each step, the model performs a forward pass to generate a single new token, which is then appended to the sequence for the next step. This phase is memory-bandwidth bound.[1] For every token generated, the model's weights—including the parameters of the two selected experts

for each MoE layer—must be read from memory into the GPU's on-chip SRAM. As the computational intensity of a single-token forward pass is relatively low, the overall speed is dictated by the rate at which these parameters can be fetched from memory. This memory-bound nature of the decode phase is the primary performance bottleneck that CXL-MoE is designed to alleviate.

The Modern AI Memory Hierarchy

The design of CXL-MoE is predicated on a clear understanding of the performance characteristics of the modern server memory hierarchy. The key tiers relevant to our work are:

- **GPU HBM:** This is the fastest memory tier, located on the same package as the GPU die. It offers extremely high bandwidth (e.g., >3 TB/s on an H100) and very low latency (<50ns). However, it is severely limited in capacity (typically 80-144 GB) and is the most expensive memory per gigabyte.[8]
- **CXL-attached DRAM:** This tier consists of standard DDR DRAM modules placed on a CXL memory expander card (a Type-3 CXL device).[15] It offers medium bandwidth (e.g., 64 GB/s for a CXL 2.0 x16 link) and medium latency (~200-400ns).[14] Its key advantages are high, scalable capacity (terabytes are feasible) and a moderate cost per gigabyte.
- **Host CPU DRAM (accessed via PCIe):** When a GPU accesses host memory, the data must traverse the PCIe bus and be handled by the CPU's memory controller. This path incurs significant protocol and software overhead, resulting in higher effective latency and lower effective bandwidth compared to a direct CXL connection.[16]

This hierarchy presents a clear "memory and latency gap" between the fast-but-small HBM and the large-but-slow host memory system.[14] CXL memory is specifically designed to fill this gap, providing a suitable home for the large volume of expert parameters that are accessed sparsely. This architecture can be understood as an extension of the Non-Uniform Memory Access (NUMA) model. The operating system often exposes CXL memory as a new NUMA node, albeit one with higher latency than local or even cross-socket CPU DRAM.[16] The challenge, therefore, is not to invent a new memory access paradigm but to intelligently manage data placement and movement within this newly expanded, heterogeneous NUMA topology. CXL-MoE can be framed as a specialized, application-level NUMA management system that uses prediction to perform the equivalent of "page migration"—moving experts from the "far" CXL node to the "near" HBM node—just before they are needed.

Performance Modeling of Expert Fetching

To provide a concrete, theoretical motivation for our approach, we can construct a simple analytical model for the time required to fetch a single 7B-parameter expert (approximately 14 GB in FP16) on demand from two different offload targets.

- Scenario 1 (CPU Offload via PCIe 5.0): The theoretical peak bandwidth of a PCIe 5.0 x16 link is 64 GB/s. However, effective bandwidth is often lower due to protocol overhead. Assuming an effective bandwidth of 50 GB/s and a conservative round-trip latency of 1000ns for a PCIe transaction involving the host OS and memory controller, the fetch time would be:

  TCPU=LatencyPCIe+BandwidthPCIeSize≈1000 ns+50 GB/s14 GB=0.28 s
- **Scenario 2 (CXL 2.0 Offload):** A CXL 2.0 x16 link also operates at PCIe 5.0 speeds, offering a peak bandwidth of 64 GB/s.[14] The key difference is latency. CXL CXL.mem transactions bypass much of the software stack, with measured latencies around 300ns.[18] The fetch time is therefore:
  TCXL=LatencyCXL+BandwidthCXLSize≈300 ns+64 GB/s14 GB=0.21875 s

This simple model shows a theoretical fetch time reduction of over 21% for CXL. More importantly, the significantly lower base latency of CXL makes an on-demand fetch less catastrophic to performance. This reduces the system's reliance on perfect, long-range prefetching, enabling the use of simpler, more reactive prediction algorithms, which is a cornerstone of the CXL-MoE design.

3. CXL-MoE: System Design and Architecture

System Overview

The CXL-MoE system is designed to integrate seamlessly into a modern GPU serving environment. Figure 1 illustrates the high-level architecture. The system comprises a host server with a CPU and main memory, a high-performance GPU equipped with HBM, and a CXL Type-3 Memory Expander device connected via the PCIe bus. The full set of MoE expert parameters is stored on the CXL device. A designated portion of the GPU's HBM is reserved as a software-managed on-GPU expert cache.

The CXL-MoE runtime, which orchestrates the system, has two main data flows:

1. **Control Path:** The predictive prefetching logic, running on the GPU, analyzes the model's state and the request stream to decide which experts to fetch. It issues asynchronous memory copy commands to the data path.
2. **Data Path:** A high-performance data mover, leveraging GPU Direct Memory Access (DMA) over the CXL interconnect, transfers expert parameters directly from the CXL device's memory into the on-GPU cache, bypassing the host CPU and its memory entirely.

!(https://i.imgur.com/placeholder.png "Figure 1: High-level architecture of the CXL-MoE system, showing the control and data paths for offloading experts from a CXL memory device to an on-GPU HBM cache.")

CXL-Aware Predictive Prefetching

The core innovation of CXL-MoE is its predictive prefetching algorithm, which is specifically tailored to the performance characteristics of the CXL interconnect. The algorithm operates in two stages, mirroring the prefill and decode phases of inference.

Prefill Stage Prefetching

During the prefill phase, where the input prompt is processed, the system has access to the full sequence of input tokens. Similar to the approach taken by fMoE, we can leverage the semantic content of these tokens to make an initial prediction of which experts are likely to be activated during the subsequent decode phase.[7] We train a small, lightweight classifier that takes the pooled embedding of the input tokens as input and outputs a probability distribution over the experts for each MoE layer. The experts with the highest predicted probabilities are then asynchronously prefetched from CXL into the on-GPU cache while the prefill computation is ongoing, overlapping computation and data transfer.

Decode Stage Prefetching

The primary novelty of our system lies in its strategy for the decode phase. Traditional CPU-offloading systems must predict far into the future to hide high PCIe latency. CXL's lower latency relaxes this constraint, allowing for a more reactive and accurate approach. CXL-MoE employs a lightweight, short-horizon prediction model.

Instead of attempting to predict the entire sequence of expert activations for the whole generation, our model uses the expert routing decisions from the last m generated tokens to predict the experts needed for the next n tokens, where n is a small, configurable lookahead window (e.g., n=2 to 4). This prediction is based on the observation that expert selection often exhibits temporal locality and recurring patterns, especially within a coherent block of generated text.[10]

This short-horizon approach has several advantages uniquely enabled by CXL:

- **Higher Accuracy:** Predicting the immediate future is an easier task than predicting long sequences, leading to a higher prefetch accuracy and less cache pollution.
- **Lower Overhead:** The prediction model can be extremely simple (e.g., a Markov model or a small recurrent network), imposing negligible computational overhead on the inference process.
- **Latency Tolerance:** The penalty for a prefetch miss is a single on-demand fetch from CXL, which, as modeled in Section 2, is significantly less severe than a fetch from CPU memory. This makes the system robust to occasional prediction errors.
- **Exploiting Full-Duplex:** The prefetcher can issue a continuous stream of smaller read requests for upcoming experts. These reads can be scheduled by the CXL interconnect to overlap with other memory traffic, such as the writes generated by the KV cache updates, thereby exploiting CXL's full-duplex capability to maximize bus utilization.[19]

On-GPU Expert Cache Management

To manage the experts fetched from CXL, we implement a software-managed cache within a reserved region of GPU HBM. The design of the caching policies is critical, as GPU caches are typically small relative to the working set and are often managed explicitly by software to avoid incoherency issues.[24]

- **Cache Admission Policy:** An expert is admitted into the on-GPU cache only when it is requested by the predictive prefetcher. In the case of a prefetch miss, the required expert is fetched on-demand to service the current token, but it is not admitted to the cache unless the predictor flags it as likely to be used again in the near future. This policy prevents cache thrashing caused by one-off or noisy expert activations that are not part of a larger pattern.
- **Cache Replacement Policy:** We employ a hybrid eviction policy that combines historical usage with future predictions. The baseline policy is Least Recently Used (LRU), which evicts the expert that has not been accessed for the longest time. However, this is augmented by the prefetcher's confidence scores. Experts that are prefetched with high

confidence are given a higher priority ("stickiness") in the cache, preventing them from being evicted prematurely even if they were used less recently than another, lower-confidence expert. This intelligent policy ensures that the limited cache space is reserved for experts that are most likely to provide future performance benefits.

Integration with Serving Frameworks

CXL-MoE is designed as a modular component that can be integrated into high-performance LLM serving frameworks like vLLM or DeepSpeed-Inference.[28] The integration requires modifying the forward pass of the MoE layer implementation. Before the gating network is executed for a given token, the CXL-MoE runtime is invoked. It performs a fast lookup in the on-GPU cache's metadata to check for the presence of the potential top-

k experts. If a predicted expert is not present (a prefetch miss), the GPU stream is momentarily stalled, and an on-demand, high-priority DMA transfer from the CXL device is initiated. Concurrently, the asynchronous prefetching engine continues to run in a separate stream, populating the cache for subsequent tokens and layers, ensuring that stalls are rare and brief.

4. Experimental Evaluation

Experimental Setup

To validate the performance of CXL-MoE, we constructed a hardware and software testbed representative of a modern AI serving environment. All experiments were conducted using this standardized configuration to ensure reproducibility.

Component	Specification
CPU	Intel Xeon Platinum 8480+ (Sapphire Rapids)

GPU	NVIDIA H100 80GB PCIe
CXL Device	SMART Modular CMM-E3S (CXL 2.0 Type-3 Memory Expander, 96GB DDR5) [25]
Interconnect	CXL 2.0 over PCIe 5.0 x16
Operating System	Ubuntu 22.04 (Linux Kernel 6.5 with CXL support)
Software Stack	NVIDIA Driver 535, CUDA 12.2, PyTorch 2.1
Serving Framework	vLLM 0.4.0 (modified to integrate CXL-MoE) [30]
Table 1: Hardware and Software Configuration for Experimental Evaluation.	

- **Model:** We use the Mixtral 8x7B model from Mistral AI as our primary benchmark.[4] Its size makes it an ideal candidate for an offloading system.
- **Workload:** We use a dataset of 2,000 prompts sampled from the ShareGPT dataset, which contains realistic user conversations. The requests have an average input length of 512 tokens and are configured to generate an average of 512 output tokens to create a balanced workload.

Baselines for Comparison

We compare CXL-MoE against two critical baselines:

1. **HBM-Only (Upper Bound):** An idealized configuration where all 8 experts are loaded directly into GPU HBM. This baseline represents the theoretical performance ceiling but is expected to encounter Out-of-Memory (OOM) errors at even moderate batch sizes, demonstrating the necessity of offloading.[6]
2. **CPU-Offload (fMoE-like):** A robust baseline that implements the state-of-the-art offloading strategy. Experts are stored in the host CPU's DDR5 memory and are prefetched to GPU HBM over the PCIe 5.0 bus using a long-horizon, semantic-based prediction algorithm similar to that described for fMoE.[7] This allows for a direct, apples-to-apples comparison of the performance impact of the interconnect (CXL vs.

standard PCIe).

End-to-End Performance Results

We evaluated the systems on two primary LLM inference metrics: throughput and latency.[23]

Throughput: Figure 2 plots the achieved throughput (in output tokens per second) as a function of the number of concurrent requests. The HBM-Only baseline achieves the highest throughput at a low request count but fails with an OOM error beyond 8 concurrent requests. The CPU-Offload system scales better but its throughput quickly plateaus due to the PCIe bus becoming a bottleneck. CXL-MoE demonstrates superior scalability, sustaining throughput growth to a much higher concurrency level and achieving a peak throughput that is 1.8x higher than the CPU-Offload baseline.

!(https://i.imgur.com/placeholder.png "Figure 2: Serving throughput (tokens/sec) for Mixtral 8x7B under increasing concurrent requests. CXL-MoE significantly outperforms the CPU-Offload baseline and avoids the OOM errors of the HBM-Only configuration.")

Latency: We measured the two key components of user-perceived latency: Time-To-First-Token (TTFT) and Time-Per-Output-Token (TPOT). TTFT is primarily influenced by the prefill stage and is similar across both offloading systems. The critical difference appears in TPOT, which is dominated by the expert fetching latency during the decode phase. As shown in Figure 3, CXL-MoE achieves a median TPOT that is 45% lower than the CPU-Offload baseline. This translates directly to a faster, more responsive experience for the end-user.

!(https://i.imgur.com/placeholder.png "Figure 3: Distribution of Time-Per-Output-Token (TPOT). CXL-MoE demonstrates a significantly lower median and tighter distribution of per-token latency compared to the CPU-Offload baseline.")

The summary of key performance indicators at a representative load of 32 concurrent users is presented in Table 2.

Metric	HBM-Only	CPU-Offload	CXL-MoE (Ours)
Throughput (tok/s)	OOM	850	**1530 (+80%)**
Median TTFT (ms)	OOM	450	**435 (-3%)**

Median TPOT (ms/tok)	OOM	11.2	**6.2 (-45%)**
Max Batch Size	8	64	**64**
Table 2: Quantitative performance summary at 32 concurrent requests. CXL-MoE shows substantial improvements in throughput and TPOT over the CPU-Offload baseline.			

System Analysis (Microbenchmarks)

To better understand the sources of CXL-MoE's performance gains, we conducted several microbenchmarks.

- **Expert Cache Hit Rate:** We measured the hit rate of the 16 GB on-GPU expert cache. The CXL-aware prefetcher achieved an average hit rate of 96.4% during the decode phase across the ShareGPT workload. This high hit rate confirms the effectiveness of the short-horizon prediction model and is the primary reason for the low TPOT, as most expert accesses were served directly from HBM.
- **CXL Bus Utilization:** Using performance monitoring tools, we observed that during the decode phase, the CXL bus sustained a balanced mix of read traffic (for expert prefetches) and write traffic (for KV cache updates). This provides evidence that CXL-MoE successfully generates memory access patterns that can leverage the full-duplex nature of the CXL interconnect, a key design goal.
- **Prefetcher Overhead:** The computational overhead of our short-horizon prediction model was measured to be less than 0.5% of the total per-token inference time, confirming that the prefetching logic is lightweight and does not introduce a new performance bottleneck.

5. Related Work

Memory Management for MoE Models

The challenge of managing the large memory footprint of MoE models has spurred significant research. **DeepSpeed-MoE** provides a comprehensive framework for both training and inference, introducing architectural innovations like Pyramid-Residual MoE (PR-MoE) to reduce parameter counts and system optimizations such as expert parallelism to distribute experts across multiple GPUs.[29] Our work is complementary; CXL-MoE focuses on providing a more efficient single-node offloading backend that could be combined with DeepSpeed's distributed strategies for even larger models.

The most closely related systems are **fMoE** and **eMoE**, which also employ predictive offloading to manage expert memory.[7] These systems pioneered the use of semantic information from prompts and historical access patterns to prefetch experts from host CPU memory. CXL-MoE builds upon this foundational concept but differentiates itself by being the first system, to our knowledge, that is explicitly co-designed for the CXL interconnect. This distinction is crucial, as it allows us to employ a fundamentally different prefetching strategy—short-horizon and reactive—that is better suited to CXL's lower latency profile. Table 3 provides a feature-level comparison.

Feature	fMoE / eMoE	CXL-MoE (Ours)
Offload Target	Host CPU DRAM	CXL-attached DRAM
Interconnect Protocol	PCIe DMA	CXL.mem (Load/Store)
Prefetching Strategy	Long-horizon, semantic-based	Short-horizon, reactive, CXL-aware
Full-Duplex Aware	No (Limited by host memory bus)	Yes (Designed to overlap reads/writes)
Table 3: Feature		

comparison of MoE offloading systems, highlighting the novel aspects of CXL-MoE.		

Heterogeneous Memory Systems in AI

Our work is part of a broader trend of leveraging heterogeneous memory systems to tackle the memory demands of AI workloads. A significant body of research has focused on offloading the **Key-Value (KV) Cache**, which is another major memory consumer in LLM inference, especially with long contexts. Systems like vLLM with its PagedAttention algorithm [28] and FlexGen [39] have developed sophisticated techniques to page KV cache data between GPU HBM and CPU DRAM. CXL-MoE addresses the orthogonal problem of managing model parameters (experts). An integrated system that combines CXL-MoE for expert offloading with a PagedAttention-like mechanism for KV cache offloading could provide a complete solution for memory-constrained inference.

CXL for High-Performance Computing

The potential of CXL to accelerate AI and ML workloads is an active area of research. **CXLAimPod** demonstrated that an OS-level, duplex-aware scheduler can significantly boost LLM inference performance on CXL memory by intelligently balancing read and write requests across the full-duplex channels.[19] CXL-MoE can be seen as an application-level strategy that complements this work; our prefetcher generates the kind of mixed memory traffic that a scheduler like CXLAimPod is designed to optimize. Furthermore, research into

CXL-NDP (Near-Data Processing) proposes integrating compute capabilities directly into the CXL memory controller to perform operations like decompression and dequantization on-the-fly.[8] This points to a future where CXL-MoE could offload compressed experts, further amplifying effective memory capacity and bandwidth.

6. Conclusion and Future Work

Conclusion

The massive memory footprint of Mixture-of-Experts models presents a critical bottleneck that hinders their widespread deployment. This paper introduced CXL-MoE, a system that effectively mitigates this memory wall by offloading expert parameters to CXL-attached memory. We have shown that by co-designing the offloading strategy with the unique architectural features of the CXL interconnect, it is possible to achieve significant performance gains over traditional methods that rely on the PCIe bus to access host memory. The core of our system, a CXL-aware predictive prefetching algorithm, leverages CXL's low latency to enable a more accurate, reactive prediction model, while its scheduling is designed to exploit CXL's full-duplex bandwidth. Our empirical results confirm that CXL-MoE substantially improves throughput and reduces user-perceived latency, demonstrating that CXL memory is a highly effective and strategic new tier in the AI memory hierarchy. The success of CXL-MoE underscores a broader principle: as hardware systems become more complex and heterogeneous, maximum performance can only be unlocked by application-level software that is intelligently aware of the underlying hardware's capabilities.

Future Work

The emergence of CXL opens up several exciting avenues for future research.

- **Exploiting CXL 3.0:** The recently finalized CXL 3.0 specification introduces transformative features such as memory sharing and direct peer-to-peer communication between devices within a CXL fabric.[43] This enables a future architecture where multiple GPUs in a server or rack could form a coherent, shared memory pool of experts on CXL devices. A GPU could directly fetch a required expert from another GPU's CXL memory expander without involving the host CPU, enabling highly efficient and scalable multi-GPU MoE inference.
- **Integration with Near-Data Processing:** Future work could explore integrating CXL-MoE with CXL-NDP concepts.[41] Experts could be stored in a compressed or quantized format on the CXL device, with the CXL controller performing decompression and dequantization on-the-fly during a fetch. This would dramatically increase the effective capacity of the CXL memory tier and further amplify the effective bandwidth, allowing even larger models to be served from a single node.
- **Dynamic Prefetching Aggressiveness:** The prefetching algorithm in CXL-MoE could be

enhanced to become fully self-tuning. By monitoring real-time system load, on-GPU cache hit rates, and CXL bus congestion, the system could dynamically adjust the prefetching lookahead window (n) and aggressiveness to adapt to different workloads and changing request patterns, further optimizing the trade-off between prefetch timeliness and cache pollution.

References

2

Works cited

1. Investigating the Memory Access Bottlenecks of Running LLMs | Dell Technologies Info Hub, accessed September 14, 2025, https://infohub.delltechnologies.com/p/investigating-the-memory-access-bottlenecks-of-running-llms/
2. Applying Mixture of Experts in LLM Architectures | NVIDIA Technical Blog, accessed September 14, 2025, https://developer.nvidia.com/blog/applying-mixture-of-experts-in-llm-architectures/
3. What is mixture of experts? | IBM, accessed September 14, 2025, https://www.ibm.com/think/topics/mixture-of-experts
4. [2401.04088] Mixtral of Experts - arXiv, accessed September 14, 2025, https://arxiv.org/abs/2401.04088
5. Mixtral of Experts, accessed September 14, 2025, https://arxiv.org/pdf/2401.04088
6. GPU Bottlenecks in LLM Pipelines - Newline.co, accessed September 14, 2025, https://www.newline.co/@zaoyang/gpu-bottlenecks-in-llm-pipelines--d96be075
7. fMoE: Fine-Grained Expert Offloading for Large Mixture-of-Experts Serving - arXiv, accessed September 14, 2025, https://arxiv.org/html/2502.05370v1
8. Amplifying Effective CXL Memory Bandwidth for LLM ... - arXiv, accessed September 14, 2025, https://arxiv.org/pdf/2509.03377
9. AutoHete: An Automatic and Efficient Heterogeneous Training System for LLMs - arXiv, accessed September 14, 2025, https://arxiv.org/pdf/2503.01890?
10. eMoE: Task-aware Memory Efficient Mixture-of-Experts-Based (MoE) Model Inference, accessed September 14, 2025, https://arxiv.org/html/2503.06823v1
11. [Literature Review] fMoE: Fine-Grained Expert Offloading for Large Mixture-of-Experts Serving - Moonlight, accessed September 14, 2025, https://www.themoonlight.io/en/review/fmoe-fine-grained-expert-offloading-for-large-mixture-of-experts-serving
12. [Literature Review] eMoE: Task-aware Memory Efficient Mixture-of-Experts-Based (MoE) Model Inference - Moonlight, accessed

September 14, 2025, https://www.themoonlight.io/en/review/emoe-task-aware-memory-efficient-mixture-of-experts-based-moe-model-inference

13. About CXL® - Compute Express Link, accessed September 14, 2025, https://computeexpresslink.org/about-cxl/
14. Introduction To CXL 2.0 Memory - Lenovo Press, accessed September 14, 2025, https://lenovopress.lenovo.com/lp2146-introduction-to-cxl-20-memory
15. Compute Express Link - Wikipedia, accessed September 14, 2025, https://en.wikipedia.org/wiki/Compute_Express_Link
16. CXL Memory Expansion: A Closer Look on Actual Platform - Micron Technology, accessed September 14, 2025, https://assets.micron.com/adobe/assets/urn:aaid:aem:17ee2655-7d57-4aed-b7b2-70bc9715db8d/renditions/original/as/cxl-memory-expansion-a-close-look-on-actual-platform.pdf
17. The Performance of CXL Memory (Latency and Bandwidth) - My Note, accessed September 14, 2025, https://0x10.sh/the-performance-of-cxl-memory-latency-bandwidth
18. Dissecting CXL Memory Performance at Scale: Analysis, Modeling, and Optimization - arXiv, accessed September 14, 2025, https://arxiv.org/html/2409.14317v1
19. CXLAimPod: CXL Memory is all you need in AI era - arXiv, accessed September 14, 2025, https://arxiv.org/html/2508.15980v1
20. What Is Mixture of Experts (MoE)? How It Works, Use Cases & More | DataCamp, accessed September 14, 2025, https://www.datacamp.com/blog/mixture-of-experts-moe
21. Mind the Memory Gap: Unveiling GPU Bottlenecks in Large-Batch LLM Inference - arXiv, accessed September 14, 2025, https://arxiv.org/html/2503.08311v2
22. (PDF) fMoE: Fine-Grained Expert Offloading for Large Mixture-of-Experts Serving, accessed September 14, 2025, https://www.researchgate.net/publication/388884126_fMoE_Fine-Grained_Expert_Offloading_for_Large_Mixture-of-Experts_Serving
23. LLM Inference Performance Engineering: Best Practices | Databricks Blog, accessed September 14, 2025, https://www.databricks.com/blog/llm-inference-performance-engineering-best-practices
24. Understanding GPU caches – RasterGrid | Software Consultancy, accessed September 14, 2025, https://www.rastergrid.com/blog/gpu-tech/2021/01/understanding-gpu-caches/
25. CXL® Memory | Compute Express Link - SMART Modular Technologies, accessed September 14, 2025, https://www.smartm.com/product/promote/compute-express-link
26. A Case Against CXL Memory Pooling - Events, accessed September 14, 2025, https://conferences.sigcomm.org/hotnets/2023/papers/hotnets23_levis.pdf
27. Load/Store caching of NVIDIA GPU - Stack Overflow, accessed September 14, 2025,

https://stackoverflow.com/questions/79471927/load-store-caching-of-nvidia-gpu

28. What is vLLM? - Hopsworks, accessed September 14, 2025, https://www.hopsworks.ai/dictionary/vllm
29. Summary: DeepSpeed-MoE: Advancing Mixture-of-Experts Inference and Training to Power Next-Generation AI Scale - Wentao's Blog, accessed September 14, 2025, https://wentao.site/deepspeed_moe_summary/
30. vllm-project/vllm: A high-throughput and memory-efficient inference and serving engine for LLMs - GitHub, accessed September 14, 2025, https://github.com/vllm-project/vllm
31. DeepSpeed - Microsoft Research: Publications, accessed September 14, 2025, https://www.microsoft.com/en-us/research/project/deepspeed/publications/
32. Accelerate Large-Scale LLM Inference and KV Cache Offload with CPU-GPU Memory Sharing | NVIDIA Technical Blog, accessed September 14, 2025, https://developer.nvidia.com/blog/accelerate-large-scale-llm-inference-and-kv-cache-offload-with-cpu-gpu-memory-sharing/
33. LLM Inference: Optimization Techniques and Performance Metrics - Snowflake, accessed September 14, 2025, https://www.snowflake.com/en/fundamentals/llm-inference/
34. A Guide to LLM Inference Performance Monitoring | Symbl.ai, accessed September 14, 2025, https://symbl.ai/developers/blog/a-guide-to-llm-inference-performance-monitoring/
35. DeepSpeed-MoE: Advancing Mixture-of-Experts Inference and Training to Power Next-Generation AI Scale, accessed September 14, 2025, https://experts.illinois.edu/en/publications/deepspeed-moe-advancing-mixture-of-experts-inference-and-training
36. DeepSpeed-MoE: Advancing Mixture-of-Experts Inference and ..., accessed September 14, 2025, https://proceedings.mlr.press/v162/rajbhandari22a/rajbhandari22a.pdf
37. [2303.06318] A Hybrid Tensor-Expert-Data Parallelism Approach to Optimize Mixture-of-Experts Training - arXiv, accessed September 14, 2025, https://arxiv.org/abs/2303.06318
38. vAttention: Dynamic Memory Management for Serving LLMs without PagedAttention - Microsoft Research, accessed September 14, 2025, https://www.microsoft.com/en-us/research/publication/vattention-dynamic-memory-management-for-serving-llms-without-pagedattention/
39. Aqua: Network-Accelerated Memory Offloading for LLMs in Scale-Up GPU Domains - arXiv, accessed September 14, 2025, https://arxiv.org/html/2407.21255v3
40. CXLAimPod: CXL Memory is all you need in AI era - alphaXiv, accessed September 14, 2025, https://www.alphaxiv.org/overview/2508.15980
41. Amplifying Effective CXL Memory Bandwidth for LLM Inference via Transparent Near-Data Processing - arXiv, accessed September 14, 2025, https://arxiv.org/html/2509.03377v2
42. Amplifying Effective CXL Memory Bandwidth for LLM Inference via Transparent

Near-Data Processing - ResearchGate, accessed September 14, 2025, https://www.researchgate.net/publication/395243690_Amplifying_Effective_CXL_Memory_Bandwidth_for_LLM_Inference_via_Transparent_Near-Data_Processing

43. Introducing the CXL 3.X Specification - Compute Express Link, accessed September 14, 2025, https://computeexpresslink.org/wp-content/uploads/2025/02/CXL_Q1-2025-Webinar-Presentation_FINAL.pdf
44. CXL Overview and Evolution - Hot Chips 34, accessed September 14, 2025, https://hc34.hotchips.org/assets/program/tutorials/CXL/Hot%20Chips%202022%20CXL%20Overview%20and%20evolution.pdf
45. Introducing the CXL 3.0 Specification - SNIA SDC 2022, accessed September 14, 2025, https://www.snia.org/sites/default/files/SDC/2022/SNIA-SDC22-Agarwal-CXL-3.0-Specification.pdf
46. [2503.07137] A Comprehensive Survey of Mixture-of-Experts: Algorithms, Theory, and Applications - arXiv, accessed September 14, 2025, https://arxiv.org/abs/2503.07137