A Simulator for LLM Inference Systems Exploiting CXL Memory Pools

Jinghan Huang, *Member, IEEE*, Hongkun Zeng, Mike Montano, Jaehong Cho, *Graduate Student Member, IEEE*, Hyunmin Choi, *Graduate Student Member, IEEE*, Jinin So, *Member, IEEE*, Junhyeok Im, Handeok Lee, Jongse Park, *Senior Member, IEEE*, and Nam Sung Kim, *Fellow, IEEE*

***Abstract*—The cost of large language model (LLM) inference is heavily tied to the immense memory capacity required to store model parameters—a challenge exacerbated by current systems that replicate these parameters across multiple servers to scale. To tackle this challenge, we may consider LLM inference systems exploiting a CXL memory pool to store and share model parameters across multiple servers, while keeping server-specific activations and KV-caches within each server's local memory. Motivated by the lack of existing simulators for such systems, we first develop a simulator that extends LLMServingSim and integrates it with NS-3 and Mess to accurately model the CXL switch and the memory modules within the CXL pool, respectively. Second, we validate this simulator against a system connecting an industry CXL memory pool to two servers running two instances of the same LLM; across separate evaluations for OPT-1.6B and OPT-6.7B, the simulator demonstrates a ≈3% error in inference latency. Lastly, using our simulator, we uncover a significant sensitivity between inference latency and the interconnect bandwidth of the CXL memory pool, demonstrating that adopting an emerging optical interconnect over the traditional electrical counterpart reduces inference latency by 67.9% and 64.4% for OPT-1.6B and OPT-6.7B, respectively.**

***Index Terms*—Compute Express Link (CXL), memory pooling, CXL switch fabric, large language model (LLM) inference, simulation.**

I. CXL Memory Pools for LLM Inference Systems

THE widespread deployment of Large Language Models (LLMs) is primarily limited by high infrastructure costs. Specifically, these models require immense memory capacity just to store their massive number of parameters. Current inference architectures worsen this problem; while they use multiple servers to meet strict Service Level Objectives (SLOs) for latency and throughput, they do so by replicating these parameters across these servers [1]. Coupled with rising memory prices, this constant replication leads to massive capital expenditures for hyperscalers.

To reduce the capital expenditures, such systems may store parameters in (1) local storage provisioned for each server or (2) remote memory shared by multiple servers, while keeping intermediate states—such as activations and Key-Value (KV) caches—in local memory. However, they suffer from considerable performance degradation and thus SLO violation, because both local storage and remote memory provide servers with high latency and low bandwidth. Specifically, accessing local storage is bottlenecked by the OS file system and block I/O stack. While aggressive prefetching can be used to hide the high latency, it requires large buffers in local memory (e.g. , ≈10 GB for a 530B model) [2], [3]. This reduces the memory capacity available for storing the intermediate states, thereby constraining the batch size. In contrast, accessing remote memory through RDMA can be ≈ 10× faster than accessing local storage [4]. Nonetheless, accessing shared remote memory implemented with RDMA is bottlenecked by the OS virtual memory system—where servers must trigger page faults to swap pages between local and remote memory—and network I/O stack.

Compute Express Link (CXL) has emerged as a promising technology to build shared remote memory, as it provides hardware-managed cache coherency and byte-addressable access to memory connected to the PCIe physical layer at low latency. Therefore, shared remote memory implemented with CXL—also referred to as a CXL memory pool in this paper—permits multiple servers to directly get parameters as if these parameters were residing in local memory, completely bypassing the OS virtual memory system and network I/O stack.

While it is promising to use a CXL memory pool for sharing parameters across servers, exploring its design space and benefits is currently hindered by a lack of high-fidelity simulation infrastructure capable of modeling these complex, disaggregated environments. To bridge this gap, we develop a simulator designed specifically for LLM inference systems using CXL memory pools. Specifically, we start with extending LLMServingSim [5] to capture the timing, address, and intensity of accesses to a memory pool by servers concurrently executing given LLMs. Subsequently, we integrate it tightly with two other popular simulators: NS-3 [6] and Mess [7]. NS-3 captures the intricate network routing and congestion dynamics of the CXL switch connecting the servers to memory modules in the memory pool. Mess models the latency behaviors of the memory modules residing within the CXL memory pool. To validate our simulator, we set up a system connecting an industry CXL memory pool to two servers, each simultaneously executing two instances of the same LLM. Through comprehensive, end-to-end evaluations running the OPT-1.6B and OPT-6.7B models, our simulator demonstrates a nominal error margin of approximately 3% in overall inference latency when compared to the physical system.

Received 13 March 2026; revised 25 April 2026; accepted 27 April 2026. Date of publication 6 May 2026; date of current version 1 June 2026. This work was supported by SRC JUMP 2.0. *(Corresponding author: Nam Sung Kim.)*

Jinghan Huang, Hongkun Zeng, Mike Montano, and Nam Sung Kim are with University of Illinois Urbana-Champaign, Urbana IL 61801 USA (e-mail: jinghan4@illinois.edu; hzeng21@illinois.edu; montano3@illinois.edu; nskim@illinois.edu).

Jaehong Cho, Hyunmin Choi, and Jongse Park are with School of Computing, Korea Advanced Institute of Science and Technology, Daejeon 34141, South Korea (e-mail: jhcho@casys.kaist.ac.kr; hmchoi@casys.kaist.ac.kr; jspark@casys.kaist.ac.kr).

Jinin So, Junhyeok Im, and Handeok Lee are with Samsung Electronics Corporation, Suwon-si 18448, South Korea (e-mail: jinin.so@samsung.com; junhyeok.im@samsung.com; handeok.lee@samsung.com).

Digital Object Identifier 10.1109/LCA.2026.3691007

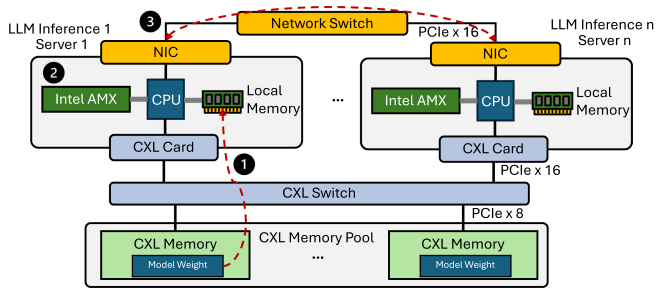

Fig. 1. CXL memory pooling for multi-instance LLM inference. Each inference server runs the same model and accesses shared read-only model parameters from a CXL memory pool through a CXL switch.

Equipped with this validated simulator, we explore the design space of a memory pool focusing on the sensitivity between end-to-end inference latency and the physical interconnect bandwidth of the memory pool. As models scale, the electrical link implementing PCIe/CXL fabrics become a severe bottleneck for token generation throughput. Modeling an emerging high-bandwidth optical link, our simulator reveals dramatic performance improvements. Specifically, we demonstrate that adopting a CXL fabric based on the optical link decreases total inference latency by 67.9% for OPT-1.6B and 64.4% for OPT-6.7B, effectively charting a highly optimized, high-performance trajectory for the next generation of scalable LLM infrastructure.

II. CXL-Based System Architecture and Simulation Model

A. Multi-Instance LLM Inference Architecture With CXL Memory Pooling

Fig. 1 illustrates our target deployment: a cluster of inference servers connected to a shared CXL memory pool through a CXL switch. Each server is equipped with Intel Advanced Matrix Extensions (AMX), reflecting a realistic modern CPU inference stack [8]. The design offloads read-only LLM model parameters into pooled CXL-attached memory. This allows multiple servers to directly map and share the same model parameters, eliminating the need for replication. Conversely, each server retains local memory for server-specific data (e.g. , activations and KV-cache). By streaming parameters from the shared pool on demand, the system facilitates overlapping communication (parameter fetching) with computation (model execution). This separation significantly reduces redundant memory footprints, improves utilization, and maximizes the number of concurrently deployable LLM inference instances.

At runtime, the execution of each layer follows a three-step workflow: ❶ **Parameter fetching:** The worker issues memory read requests for the next layer's parameters. These requests traverse the CXL switch and are serviced by the CXL memory modules. ❷ **Model execution:** The worker executes the layer using locally resident activations and KV-caches and the fetched model parameters. Latency is hidden by overlapping the parameter fetch of the next layer with the computation of the current layer. ❸ **Network communication:** If the model is partitioned across servers (pipeline/tensor parallelism), this step also handles synchronization of intermediate states via the network.

B. Simulator With CXL Memory Pool

We model the CXL switch and the memory modules within the CXL pool on top of LLMServingSim [5], a system-level simulation platform for large-scale LLM inference. To enable end-to-end evaluation of multiple LLM inference instances sharing LLM model parameters through a CXL memory pool, we introduce four **M**odifications (Ⓜ1–Ⓜ4). We utilize Mess for accurate bandwidth-latency analysis and NS-3 for high-fidelity queuing simulation, integrating these components via event-driven communication within LLMServingSim.

Ⓜ1 *Memory backend:* We augment the memory subsystem with Mess [7], a memory simulation tool that models bandwidth–latency behavior using pre-characterized curves derived from real hardware measurements. We integrate Mess directly with the simulator's memory controller and support both local memory and CXL memory pool. The latency calculator dynamically computes access latency based on windowed bandwidth measurements derived from observed traffic. For each memory request, Mess captures queuing delays, achieving high accuracy with substantially lower overhead than the simulator's original analytical memory model.

Ⓜ2 *Network backend:* To overcome the limitations of simple analytical communication models, we incorporate an NS-3 network backend. The simulator exports logical communication events to NS-3, which models packet-level effects, including link bandwidth, latency, congestion, and queuing dynamics under configurable topologies. By generating network parameters from cluster specifications (e.g., node counts and link bandwidth), we ensure that the serving workload is evaluated under realistic network conditions.

Ⓜ3 *CXL switch:* We introduce a dedicated CXL switch component to model the interconnect behavior between servers and the memory modules within the CXL memory pool. Built upon an NS-3 switch design, this component routes memory requests to the appropriate CXL memory modules while modeling switch traversal latency and traffic contention. The switch also handles the address translation and memory mapping, integrating seamlessly with the Mess memory backend to capture end-to-end remote memory access latency.

Ⓜ4 *Multi-instance execution:* We extend LLMServingSim from a single-instance simulator into a multi-instance framework. Specifically, we introduce an instance-configuration component in which each instance is described by its model, hardware profile, parallelism strategy, and memory capacity. A configuration builder compiles these descriptions into backend configuration files and maps instances to servers. At runtime, the simulator instantiates per-instance schedulers and local memory models, while multiple instances share a unified network model and a CXL memory pool. A global request router dispatches incoming requests to instances according to configurable policies (e.g. , round-robin or random), enabling simulation of concurrent LLM inference workloads.

III. Evaluation

This section validates our multi-instance extension to LLMServingSim and its CXL memory pool model. We compare

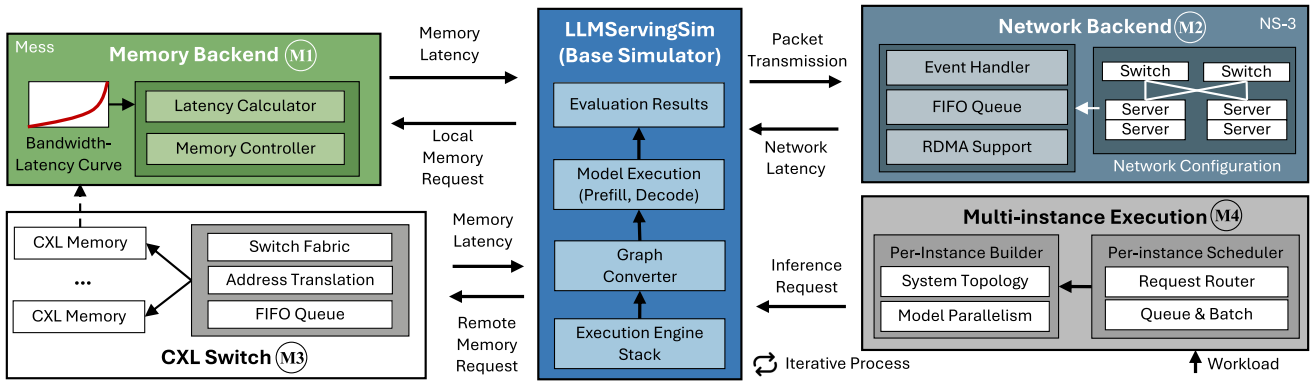

Fig. 2. LLMServingSim extensions for modeling a CXL memory pool. Our simulator integrates the Mess memory backend (M1), NS-3 network backend (M2), a CXL switch model (M3), and a multi-instance execution controller (M4).

TABLE I
EVALUATION SYSTEM CONFIGURATIONS

Component	Specification
Host CPU	Intel Xeon Platinum 8568CXL (48 cores)
Host Memory	64GB DDR5-4800 (1 channel)
OS / Kernel	Ubuntu 22.04 LTS (Linux 5.15.0)
CXL Switch	XCONN XC50256
CXL Memory	3×128GB DDR5-5200

simulated and measured latency–bandwidth curves for access to the CXL memory pool through a CXL switch under both single-host and concurrent two-host traffic, and validate end-to-end inference latency for OPT-1.6B and OPT-6.7B using input/output-length microbenchmarks on the same platform. We then use the calibrated simulator to quantify the sensitivity of multi-instance inference latency to CXL switch bandwidth when model parameters are placed in the CXL memory pool.

A. Simulator Validation Using an Industry CXL Memory Pool System

Experimental setup: Table I summarizes our evaluation platform, which currently supports only two hosts. Accordingly, we evaluate on two Intel Xeon Platinum 8568CXL (48 cores; Emerald Rapids) servers, each equipped with 64 GB DDR5-4800 DRAM and running Ubuntu 22.04 LTS with Linux 5.15.0. The hosts connect to an XCONN XC50256 CXL switch and CXL memory expanders, providing a CXL memory pool with 3×128 GB DDR5-5200 memory. To incorporate the nuances of our hardware platform—including Intel's AMX—into our compute model, we profile OPT-1.6B and OPT-6.7B inference before simulation to obtain per-operation latencies by measuring execution time of operations to be replayed during simulation. The simulator features a highly configurable and scalable architecture: cluster topology, host memory, CXL memory pool, network, per-instance model, target hardware, parallelism, weight/KV-cache placement, and scheduler policies are all set through high-level parameters. Adapting to a new hardware platform or model requires a one-time latency profile.

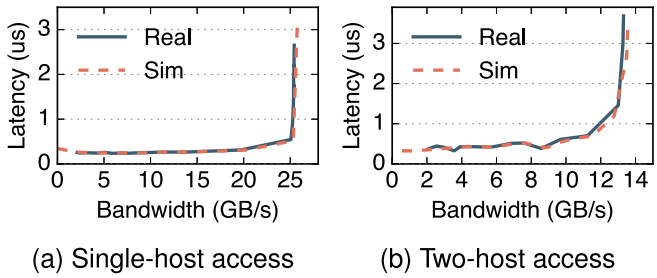

Fig. 3. Memory simulation validation. Latency–bandwidth curves from the real system and our simulator for accesses to the CXL memory pool through a CXL switch: (a) single-host access and (b) concurrent two-host access.

Validation of CXL Memory Pool Access: Fig. 3 validates our Mess memory backend and CXL switch model by comparing measured and simulated latency–bandwidth curves for single-host access and concurrent two-host access to the CXL memory pool through the switch. Under light load, the real system shows a baseline latency of 0.24μs for single-host access, which increases to 0.33 μs under concurrent two-host access due to contention. As bandwidth approaches saturation, latency increases rapidly: single-host latency rises to 2.7μs near 25 GB/s, while concurrent two-host latency reaches 3.7μs around 13 GB/s. In both cases, the simulator tracks the measured trends closely, with 4.9% error for single-host access and 10.5% error for concurrent two-host access, capturing both fixed access overheads and bandwidth-dependent queuing effects.

Validation of End-to-End Inference Latency: Fig. 4 validates end-to-end inference latency for OPT-1.6B on our platform using two microbenchmarks: (a) fixing the input length at 32 tokens while sweeping the output length, and (b) fixing the output length at 32 tokens while sweeping the input length. In Fig. 4(a), the measured latency increases from 0.5 s at 4 output tokens to 58.3 s at 512 output tokens, and the simulator tracks the same scaling trend from 0.4 s to 56.1 s, achieving 4.6% error. In Fig. 4(b), the measured latency remains nearly constant as the input length increases, ranging from 3.5 s to 3.6 s across 1–128 input tokens, and the simulator closely matches this behavior with 0.8% error. We further validate OPT-6.7B,

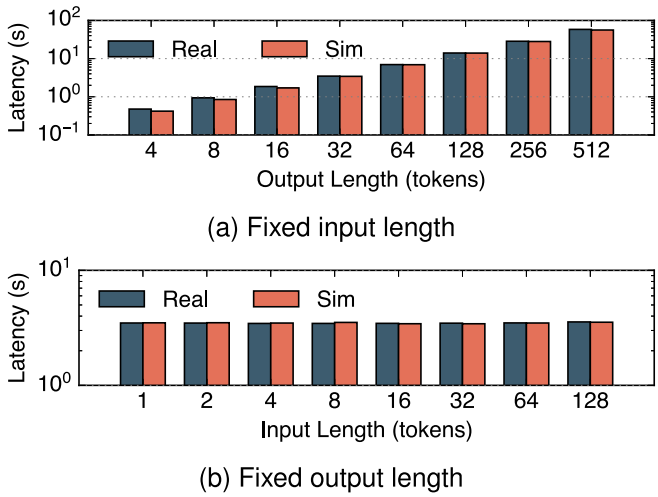

Fig. 4. End-to-end OPT-1.6B inference performance using a CXL memory pool. Simulated latency is validated against measured latency for a fixed input/output length of 32 tokens while sweeping the output/input length.

obtaining 3.1% error for the fixed-input sweep and 3.3% error for the fixed-output sweep. Comparable end-to-end LLM-serving simulators report latency errors of less than 9% (Vidur [9]) and 14.7% (LLMServingSim [5]) against measured GPU-serving baselines. Our validation error falls below both reported values.

B. Sensitivity to CXL Switch Bandwidth

After validating our modified LLMServingSim, we use it to study how multiple concurrent LLM inference instances perform under different CXL switch bandwidths. When model parameters are placed in the CXL memory pool, parameter-fetch traffic traverses the CXL switch and can become the dominant bottleneck, increasing queuing delay and reducing the effectiveness of overlapping communication with layer computation. We simulate inference with model parameters placed in the CXL memory pool (input length = 32 tokens, output length = 64 tokens) and quantify how CXL interconnect bandwidth affects end-to-end inference latency. With high-bandwidth photonic interconnects, CXL switch bandwidth can potentially scale up to 64Tbps [10]. Considering a PCIe Gen4 SSD and 400 Gbps network, the CXL memory pool can reduce end-to-end latency by up to 3.8× for local storage and 1.5× for remote memory.

Varying CXL switch bandwidth: We sweep the effective CXL switch bandwidth from 100 Gbps to 64Tbps. As shown in Fig. 5, limited interconnect bandwidth substantially increases end-to-end inference latency when fetching model parameters from the CXL memory pool. For OPT-1.6B, latency decreases from 19.6 s at 100 Gbps to 6.3 s at 64 Tbps, corresponding to a 67.9% reduction. For OPT-6.7B, latency decreases from 98.3 s at 100 Gbps to 35.0 s at 64Tbps, corresponding to a 64.4% reduction. Larger models exhibit higher absolute sensitivity to

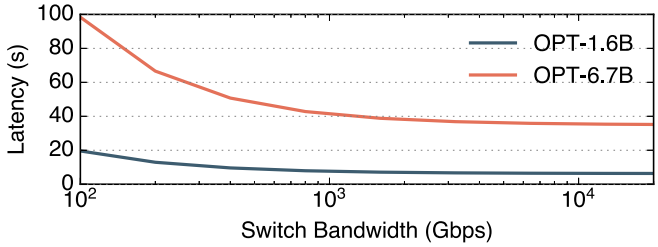

Fig. 5. Multi-instance LLM inference under varying CXL switch bandwidth, with model parameters placed in a CXL memory pool and accessed through a CXL switch (input length = 32, output length = 64).

interconnect bandwidth: moving from 100 Gbps to 64 Tbps reduces latency by 13.3 s for OPT-1.6B and 63.3 s for OPT-6.7B. CXL memory pool helps most when multiple instances share a large model with sufficient interconnect bandwidth.

IV. Conclusion

We extend LLMServingSim with Mess and NS-3 to model the CXL switch and the CXL memory modules within the CXL memory pool, and validate the simulator on a two-host platform equipped with a CXL switch and CXL memory expanders. Using our simulator, we show that increasing bandwidth from traditional electrical interconnects to emerging optical interconnects reduces inference latency by 67.9% for OPT-1.6B and 64.4% for OPT-6.7B.

References

[1] D. Crankshaw, X. Wang, G. Zhou, M. J. Franklin, J. E. Gonzalez, and I. Stoica, "Clipper: A low-latency online prediction serving system," in *Proc. 14th USENIX Conf. Networked. Syst. Des. Implementation*, 2017, pp. 613–627.

[2] Y. Sheng et al., "FlexGen: High-throughput generative inference of large language models with a single GPU," in *Proc. Int. Conf. Mach. Learn.*, 2023, pp. 31094–31116.

[3] J. Rasley, S. Rajbhandari, O. Ruwase, and Y. He, "DeepSpeed: System optimizations enable training deep learning models with over 100 billion parameters," in *Proc. 26th ACM SIGKDD Int. Conf. Knowl. Discov. Data Mining*, 2020, pp. 3505–3506.

[4] S. Ma, T. Ma, K. Chen, and Y. Wu, "A survey of storage systems in the RDMA era," *IEEE Trans. Parallel Distrib. Syst.*, vol. 33, no. 12, pp. 4395–4409, Dec. 2022.

[5] J. Cho, M. Kim, H. Choi, G. Heo, and J. Park, "LLMServingSim: A HW/SW co-simulation infrastructure for LLM inference serving at scale," in *Proc. IEEE Int. Symp. Workload Characterization*, 2024, pp. 15–29.

[6] G. F. Riley and T. R. Henderson, "The NS-3 network simulator," in *Modeling and Tools for Network Simulation*. Berlin, Germany: Springer, 2010, pp. 15–34.

[7] P. Esmaili-Dokht et al., "A mess of memory system benchmarking, simulation and application profiling," in *Proc. 57th IEEE/ACM Int. Symp. Microarchitecture*, 2024, pp. 136–152.

[8] H. Kim, N. Wang, Q. Xia, J. Huang, A. Yazdanbakhsh, and N. S. Kim, "LIA: A single-GPU LLM inference acceleration with cooperative AMX-Enabled CPU-GPU computation and CXL offloading," in *Proc. 52nd Annu. Int. Symp. Comput. Architecture*, 2025, pp. 544–558.

[9] A. Agrawal et al., "VIDUR: A large-scale simulation framework for LLM inference," in *Proc. Mach. Learn. Syst.*, 2024, vol. 6, pp. 351–366 .

[10] Z. Wu and K. Bergman, "Silicon photonic accelerated memory pooling for efficient compute resource allocation," in *Proc. IEEE Symp. High-Perform. Interconnects*, 2025, pp. 36–47.