Exploring CXL-based KV Cache Storage for LLM Serving

Yupeng Tang[1*] **Runxiang Cheng**[2*] **Ping Zhou**[3] **Tongping Liu**[3] **Fei Liu**[3] **Wei Tang**[3]
Kyoungryun Bae[3] **Jianjun Chen**[3] **Wu Xiang**[3] **Rui Shi**[3]
[1]Yale University [2]University of Illinois Urbana-Champaign [3]ByteDance

Abstract

Large language model (LLM) serving systems often store key-value (KV) cache to reduce redundant inference computations. However, storing KV cache of long-context requests under high-throughput serving demand can overwhelm GPU and host memory. Recently, Compute Express Link (CXL) emerges as a promising interconnect technology that offers low-latency host-device communication and expanded memory capacity. In this paper, we explore leveraging CXL memory to store KV cache in LLM serving. Our results show that CXL-CPU interconnect performs comparably to CPU-GPU interconnect in both data-transfer latency and bandwidth, enabling a 30% increase in batch size compared to full KV re-compute under the same SLO. Our production Return on Investment (ROI) modeling further shows that storing KV cache on CXL memory can reduce GPU requirements by up to 87%, with 7.5× higher GPU utilization for prefill, compared to full KV re-compute. Our overall results demonstrate the performance improvement and resource benefits of using CXL memory to store KV cache in LLM serving.

1 Introduction

Autoregressive large language models (LLMs) generate output tokens sequentially, where the generation of each token involves the attention computation using key-value (KV) of its preceding tokens [1]. This sequential dependency makes LLM inference both compute- and memory-intensive. LLM inference typically includes two stages: the prefill stage, where all input tokens are processed to generate the initial output token, and the decode stage, where the rest of the output tokens are generated one by one until the model generates an end-of-sequence token [2, 3, 4].

For applications such as chatbot and coding assistant, LLM serving systems aim to minimize the time to finish the prefill stage, or time to first token (TTFT). In production, service-level objective (SLO) for TTFT is typically 400ms [3]. To meet such SLO, LLM serving systems often cache the previously-computed KV data of the preceding tokens (i.e., prefix) in GPU memory, to avoid re-computing them for future requests that have the same prefix [5, 3, 6]. Storing KV cache reduces the overall computational load and significantly improves throughput by trading memory for computation.

In production chatbot applications that support large context windows, the demand for KV cache storage grows rapidly by the number of inference requests from users, which cannot be fully accommodated by the limited and expensive GPU memory [7]. Researchers thus developed techniques to offload KV cache to CPU memory, leveraging the larger CPU memory capacity to reduce GPU memory pressure [6, 8, 9]. However, as larger LLMs and support for long-context inference requests continue to emerge, the approach of storing KV cache to CPU memory is still insufficient. For example, in LLaMA-2-7B, KV cache of token in FP32 precision is 1024KB; KV cache of a single

*Co-first authors. Work done during internship at ByteDance.

Machine Learning for Systems Workshop at (NeurIPS 2024).

request with 4096 tokens (maximum context length) is 4GB [10]. The memory demand from serving many concurrent long-context requests can easily overwhelm even high-end memory servers [5, 11].

Practitioners increasingly turn to more scalable memory architectures, such as Compute Express Link (CXL) memory [12, 13, 14], to address the growing memory demands of large-scale systems. CXL expands memory capacity by connecting additional DRAM to servers via PCIe, while maintaining low-latency access. It offers a promising solution to the KV cache storage demand in LLM serving.

In this paper, We propose leveraging CXL memory for KV cache storage to improve throughput, meet SLO on TTFT, and alleviate memory pressure in LLM serving systems. This paper makes the following contributions:

- We present the first measurement of CXL-GPU interconnect and its feasibility for storing large KV cache. We show that the data-transfer latency and bandwidth on CXL-GPU interconnect is on par with CPU-GPU interconnect.
- We present our design of CXL-based KV cache storage interface and evaluate its performance improvement to LLM serving, on our platform that is the first to successfully integrate ASIC-CXL device and GPU. Our results show competitive TTFT achieved by CXL-based prefix caching.
- We examine the cost-efficiency in using CXL for KV cache storage in production via Return on Investment (ROI) modeling. Estimates show a promising reduction in GPU compute cost when using CXL for KV cache storage. We also identify promising future research directions.

2 CXL-based KV Cache Storage

We now present the design and implementation of our CXL-based KV cache storage interface for LLM serving. We also describe the hardware platform used to evaluate our design.

Design and implementation. Our goal is to develop a CXL storage interface which can be integrated into existing LLM serving systems for saving and loading KV cache of inference requests. The interface provides two external APIs to its upper-level serving system: `save` and `load`. The `save` takes a unique identifier of a token chunk as input, and copies its KV cache from GPU to CXL memory. The `load` takes a unique identifier of a token chunk as input, and finds if its KV cache exists in CXL memory, if so, copies the KV cache from CXL memory to GPU. A token chunk can consist of one or more tokens. The unique identifier of a token chunk t_i for a sequence is the hash of the content of t_i and the hash of its prefix $\langle t_0, ..., t_{i-1} \rangle$. If the KV of the current request's prefix has been computed and saved into CXL, the KV cache will be loaded from CXL and used [5].

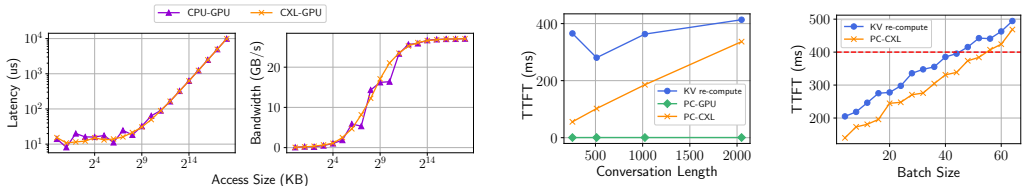
To avoid calling `save` and `load` too frequently and incurring unnecessary overhead to the upper-level serving system, `save` is called only when a request is finished so the KV cache of all the tokens for that request is saved at once; `load` is called when the prefill stage of a request begins.

We implement our design of CXL-based KV cache storage interface in gpt-fast [15], a low-latency text generation system with support on a number of widely-used inference optimizations [16, 17, 18] and open-source LLMs [10, 19]. We modify gpt-fast to support our evaluation on batched inference.

Hardware platform. Our single socket server is equipped with Intel Xeon Platinum processors [20], 1TB of 4800 MHz DDR5 memory, an NVIDIA H100 GPU with 96GB HBM, and a CXL memory expansion card with 256 GB of DDR5 memory at 4800 MHz [13]. While prior works [21, 22, 23, 24] have explored utilizing CXL for accelerators, to our knowledge, we are the first to integrate a real ASIC-CXL device and a GPU within a single inference server and evaluate it for KV cache storage.

3 Performance Evaluation

In Section 3.1, we measure the latency and bandwidth of CXL-GPU interconnect for data transfer to assess the feasibility of storing KV cache on CXL devices. In Section 3.2, we compare the TTFT of KV re-compute, prefix caching with CXL, and prefix caching with GPU, to understand if CXL-based KV cache storage can achieve similar TTFT as existing approaches for prefill requests under varying context lengths. In Section 3.3, we study the maximum batch size achieved while retaining a given SLO on TTFT between KV re-compute and prefix caching with CXL. In Section 3.4, we model the ROI of CXL-based KV cache storage in production.



(a) CPU-GPU/CXL-GPU interconnect. (b) TTFT comparison. (c) Max BS under SLO.

Fig. 1: **Experiment results.** (a) Latency and bandwidth measurements across different access sizes, CXL-GPU interconnect performs similarly as CPU-GPU interconnect. (b) TTFT comparison between KV re-compute and prefix caching with CXL or GPU. (c) Serving throughput comparison under a fixed SLO constraint (400ms).

3.1 Measurements on CXL-GPU interconnect performance

KV cache storage requires low-latency access. Although prior studies [12, 13] show that accessing CXL memory from the host CPU is over 2× slower than accessing local memory, none of their measurements involves any interaction with the GPU. In this paper, we evaluate the performance characteristics of the CXL-GPU interconnect by measuring the latency and bandwidth of copying data from CXL memory to the GPU. Transferring in the reverse direction yields similar results [25]. Since CXL memory devices are exposed to the system as NUMA nodes without CPUs by default [13], we allocate a set of host buffers on the CXL NUMA node and use `cudaMemcpyAsync` to copy data between the host buffers and GPU device buffers allocated via the CUDA API [26]. While prior work evaluate data transfer in 64 bytes [24], we evaluated on sizes ranging from 1KB to 256MB.

Figure 1(a) shows that the performance of the CXL-GPU interconnect **is unexpectedly on par with** traditional CPU-GPU memory transfers, exhibiting no significant slowdown. Latency remains low for smaller access sizes but increases exponentially once the size exceeds 64KB. Meanwhile, bandwidth increases almost linearly with data size and saturates around 4MB. This indicates that, while the CPU oversees the data transfer, the data path actually bypasses the host's local memory, flowing directly from CXL memory to GPU buffers via PCIe. Our results demonstrate that the CXL-GPU interconnect operates efficiently with minimal latency overhead, positioning it as a promising expansion for KV cache offloading [9, 27, 28] and swapping [5] in addition to CPU memory.

3.2 Evaluation on TTFT under varying input context length

Given that CXL-GPU interconnect performs nearly the same as CPU-GPU interconnect, we further study if CXL-based KV cache storage can achieve similar TTFT as existing approaches in completing the prefill stage computation for an inference request. We evaluate three approaches:

- **Full KV re-compute:** Compute KV data of all input tokens for the request with GPU.
- **Prefix caching with CXL:** Load KV cache of the prefix tokens for the request from CXL to GPU.
- **Prefix caching with GPU:** Store and use KV cache in GPU for the prefix tokens for the request.

We measure the TTFT of the aforementioned approaches on conversation requests of input length ranging from 256 to 2048 tokens from the ShareGPT-Vicuna-Unfiltered dataset [29]. We use the LLaMA-2-13B as the underlying model for our evaluation. Figure 1(b) shows the TTFT (y-axis in log-scale) achieved by the evaluated approaches for requests of varying input context length (x-axis).

Compared to the other approaches, prefix caching with GPU (denoted as "PC-GPU" in Figure 1(b)) achieves the smallest TTFT (0.44ms to 0.56ms) constantly across different input context lengths. Such performance is expected as there is no data transfer latency and computation of KV data is only needed for tokens after the prefix. This approach is an optimal baseline that is however difficult to achieve in practice due to limited memory capacity of existing GPU models and the rapidly growing demand of KV cache storage in LLM serving.

Comparing prefix caching with CXL (denoted as "PC-CXL") and KV re-compute, prefix caching with CXL performs at least as good as computing KV data on GPU from scratch. Prefix caching with CXL achieve TTFT ranging from 55ms to 336ms, with slight increase in latency as input size length grows. The close performance gap between storing prefix KV cache in CXL memory and full KV re-computation indicates that there is a potential opportunity to reduce GPU compute cost with adaptation of CXL devices for memory capacity expansion in LLM inference.

3.3 Evaluation on serving throughput while adhering SLO

By storing the KV cache of the inference request prefix in CXL memory and thus reducing re-computation during the prefill stage, we can effectively reduce the computational load on the GPU. The saved GPU compute can be re-allocated to handle a larger number of concurrent inference requests. In other words, the LLM serving system can achieve a higher serving throughput, by handling a larger batch size of inference requests using the saved GPU compute, while maintaining the same SLO on TTFT [3].

Figure 1(c) shows the TTFT achieved by KV re-compute and prefix caching with CXL under varying batch size. The horizontal red-dashed line indicates our SLO limit–the maximum TTFT that can be tolerant in production. The typical SLO is 400ms used for LLaMA-2 [30]. As shown in Figure 1(c), with KV re-compute, the evaluated serving system (§2) can handle a maximum batch size of 44 before hitting the SLO limit. On the other hand, when leveraging CXL for storing KV cache, the system can handle a maximum batch size of 57, which is a **30% increase** compared to KV re-compute. Our initial evaluation on SLO-adhering serving throughput highlights the performance benefits of utilizing CXL memory for KV cache storage, particularly in scenarios that require efficient scaling under strict latency requirements.

3.4 Cost-efficiency modeling

We apply a Return on Investment (ROI) model to estimate the cost-efficiency of CXL-based KV cache storage, compared to full KV re-compute, in our production for chatbot application (§A.1).

- **Assumption:** A GPU has the computational power of 100 TFLOP/s, a prefill request on average requires 25 TFLOP of computation, and the SLO on TTFT is 400ms (i.e., 0.4s).
- **KV re-compute (baseline):** To complete the prefill request within SLO, each request demands 62.5 TFLOP/s (25 TFLOP / 0.4s), meaning a single GPU can serve 1.6 prefill requests per second.
- **CXL-based KV cache storage:** By spending 0.1s loading the KV cache of the request prefix, we reduce the computational demand to 2.5 TFLOP (assuming the prefix accounts for 90% of the input for a request). To meet the same SLO, the remaining computation must be finished within 0.3s, requiring 8.3 TFLOP/s (2.5 TFLOP / 0.3s). In this case, a single GPU can serve 12 prefill requests, which is 7.5× more requests per second with the same amount of GPU compute.

From the estimate, we can reduce the number of GPUs needed to reach the same throughput while adhering SLO on TTFT by 87% when compared to KV re-compute. Overall, by replacing computation with CXL memory access, we reduce the overall compute demand while still meeting the same SLO, thereby resulting in significant GPU cost savings for LLM inference.

4 Conclusion and Future Work

Storing KV cache in GPU memory for LLM inference can quickly lead to memory saturation, limiting serving scalability and performance. KV cache storage on CPU memory becomes limited as model size and request context length increase. To that extent, we explore CXL memory for KV cache offloading, in which CXL offers expanded capacity with low-latency access. Our preliminary results show that CXL-CPU data transfer has similar latency and bandwidth as the CPU-GPU counterpart. In addition, CXL-based KV cache offloading provides similar performance compared to full KV re-compute on GPUs, while supporting larger workloads. Specifically, using CXL memory for KV cache storage increased the maximum batch size by 30%, while maintaining the same SLO on TTFT. Our cost-efficiency analysis further shows the potential for using CXL memory to substantially reduce the GPU compute cost for high-throughput LLM serving under SLO. Looking ahead, future work will explore the integration of CXL memory with multi-GPU systems, focusing on maintaining cache coherence across GPUs that could further enhance the scalability and efficiency of LLM inference.

Acknowledgments and Disclosure of Funding

We thank Supermicro for the hardware. We thank Ken Hu and Henry Hu from ByteDance for assisting experiment setup; Wenhui Zhang from ByteDance, for insightful suggestions to this work.

References

[1] Josh Achiam, Steven Adler, Sandhini Agarwal, Lama Ahmad, Ilge Akkaya, Florencia Leoni Aleman, Diogo Almeida, Janko Altenschmidt, Sam Altman, Shyamal Anadkat, et al. Gpt-4 technical report. *arXiv preprint arXiv:2303.08774*, 2023.

[2] Amey Agrawal, Ashish Panwar, Jayashree Mohan, Nipun Kwatra, Bhargav S Gulavani, and Ramachandran Ramjee. Taming throughput-latency tradeoff in llm inference with sarathi-serve. In *OSDI*, 2024.

[3] Yinmin Zhong, Shengyu Liu, Junda Chen, Jianbo Hu, Yibo Zhu, Xuanzhe Liu, Xin Jin, and Hao Zhang. Distserve: Disaggregating prefill and decoding for goodput-optimized large language model serving. In *OSDI*, 2024.

[4] Pratyush Patel, Esha Choukse, Chaojie Zhang, Aashaka Shah, Íñigo Goiri, Saeed Maleki, and Ricardo Bianchini. Splitwise: Efficient generative llm inference using phase splitting. In *ISCA*, 2024.

[5] Woosuk Kwon, Zhuohan Li, Siyuan Zhuang, Ying Sheng, Lianmin Zheng, Cody Hao Yu, Joseph Gonzalez, Hao Zhang, and Ion Stoica. Efficient memory management for large language model serving with pagedattention. In *SOSP*, 2023.

[6] Cunchen Hu, Heyang Huang, Junhao Hu, Jiang Xu, Xusheng Chen, Tao Xie, Chenxi Wang, Sa Wang, Yungang Bao, Ninghui Sun, et al. Memserve: Context caching for disaggregated llm serving with elastic memory pool, 2024.

[7] Xupeng Miao, Gabriele Oliaro, Zhihao Zhang, Xinhao Cheng, Hongyi Jin, Tianqi Chen, and Zhihao Jia. Towards efficient generative large language model serving: A survey from algorithms to systems. *arXiv preprint arXiv:2312.15234*, 2023.

[8] Jiayi Yao, Hanchen Li, Yuhan Liu, Siddhant Ray, Yihua Cheng, Qizheng Zhang, Kuntai Du, Shan Lu, and Junchen Jiang. Cacheblend: Fast large language model serving for rag with cached knowledge fusion. In *EuroSys*, 2024.

[9] 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 *ICML*, 2023.

[10] Hugo Touvron, Louis Martin, Kevin Stone, Peter Albert, Amjad Almahairi, Yasmine Babaei, Nikolay Bashlykov, Soumya Batra, Prajjwal Bhargava, Shruti Bhosale, et al. Llama 2: Open foundation and fine-tuned chat models. *arXiv preprint arXiv:2307.09288*, 2023.

[11] Yuhan Liu, Hanchen Li, Kuntai Du, Jiayi Yao, Yihua Cheng, Yuyang Huang, Shan Lu, Michael Maire, Henry Hoffmann, Ari Holtzman, et al. Cachegen: Fast context loading for language model applications. 2024.

[12] Yan Sun, Yifan Yuan, Zeduo Yu, Reese Kuper, Chihun Song, Jinghan Huang, Houxiang Ji, Siddharth Agarwal, Jiaqi Lou, Ipoom Jeong, et al. Demystifying cxl memory with genuine cxl-ready systems and devices. In *MICRO*, 2023.

[13] Yupeng Tang, Ping Zhou, Wenhui Zhang, Henry Hu, Qirui Yang, Hao Xiang, Tongping Liu, Jiaxin Shan, Ruoyun Huang, Cheng Zhao, et al. Exploring performance and cost optimization with asic-based cxl memory. In *EuroSys*, 2024.

[14] 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 *ASPLOS*, 2023.

[15] PyTorch Labs. GPT-fast: Simple and Efficient Pytorch-Native Transformer tTxt Generation. https://github.com/pytorch-labs/gpt-fast.git, 2024.

[16] Yaniv Leviathan, Matan Kalman, and Yossi Matias. Fast inference from transformers via speculative decoding. In *ICML*, 2023.

[17] Deepak Narayanan, Mohammad Shoeybi, Jared Casper, Patrick LeGresley, Mostofa Patwary, Vijay Korthikanti, Dmitri Vainbrand, Prethvi Kashinkunti, Julie Bernauer, Bryan Catanzaro, et al. Efficient large-scale language model training on gpu clusters using megatron-lm. In *SC*, 2021.

[18] Benoit Jacob, Skirmantas Kligys, Bo Chen, Menglong Zhu, Matthew Tang, Andrew Howard, Hartwig Adam, and Dmitry Kalenichenko. Quantization and training of neural networks for efficient integer-arithmetic-only inference. In *CVPR*, 2018.

[19] Albert Q Jiang, Alexandre Sablayrolles, Arthur Mensch, Chris Bamford, Devendra Singh Chaplot, Diego de las Casas, Florian Bressand, Gianna Lengyel, Guillaume Lample, Lucile Saulnier, et al. Mistral 7b. *arXiv preprint arXiv:2310.06825*, 2023.

[20] Intel Corporation. Intel® Xeon® Platinum Processor. https://www.intel.com/content/www/us/en/products/details/processors/xeon/scalable/platinum.html, 2024.

[21] Moiz Arif, Avinash Maurya, and M Mustafa Rafique. Accelerating performance of gpu-based workloads using cxl. In *HPDC FlexScience*, 2023.

[22] Shintaro Sano, Yosuke Bando, Kazuhiro Hiwada, Hirotsugu Kajihara, Tomoya Suzuki, Yu Nakanishi, Daisuke Taki, Akiyuki Kaneko, and Tatsuo Shiozawa. Gpu graph processing on cxl-based microsecond-latency external memory. In *SC-W*, 2023.

[23] Donghyun Gouk, Seungkwan Kang, Hanyeoreum Bae, Eojin Ryu, Sangwon Lee, Dongpyung Kim, Junhyeok Jang, and Myoungsoo Jung. Breaking barriers: Expanding gpu memory with sub-two digit nanosecond latency cxl controller. In *HotStorage*, 2024.

[24] Jie Liu, Xi Wang, Jianbo Wu, Shuangyan Yang, Jie Ren, Bhanu Shankar, and Dong Li. Exploring and evaluating real-world cxl: Use cases and system adoption. *arXiv preprint arXiv:2405.14209*, 2024.

[25] Ang Li, Shuaiwen Leon Song, Jieyang Chen, Jiajia Li, Xu Liu, Nathan R Tallent, and Kevin J Barker. Evaluating modern gpu interconnect: Pcie, nvlink, nv-sli, nvswitch and gpudirect. *IEEE Transactions on Parallel and Distributed Systems*, 2019.

[26] NVIDIA. CUDA Runtime API - Memory Management Functions. https://docs.nvidia.com/cuda/cuda-runtime-api/group__CUDART__MEMORY.html, 2024.

[27] Reza Yazdani Aminabadi, Samyam Rajbhandari, Ammar Ahmad Awan, Cheng Li, Du Li, Elton Zheng, Olatunji Ruwase, Shaden Smith, Minjia Zhang, Jeff Rasley, et al. Deepspeed-inference: enabling efficient inference of transformer models at unprecedented scale. In *SC*, 2022.

[28] Wonbeom Lee, Jungi Lee, Junghwan Seo, and Jaewoong Sim. {InfiniGen}: Efficient generative inference of large language models with dynamic {KV} cache management. In *OSDI*, 2024.

[29] anon8231489123. ShareGPT Vicuna Unfiltered Dataset. https://huggingface.co/datasets/anon8231489123/ShareGPT_Vicuna_unfiltered, 2024.

[30] Meta. Llama 2 70B: An MLPerf Inference Benchmark for Large Language Models. https://mlcommons.org/2024/03/mlperf-llama2-70b, 2024.

A Appendix

A.1 Definition of Return on Investment (ROI) Model

Table 1 provides the definition of our ROI model used in §3.4 in the main text.

Table 1: ROI model.

C_0	Avg. FLOPs needed by a prefill request in an initial request. Can be estimated as $C_0 = 2ML$, where M is the model parameters and L is the avg. sequence length.
C_1	FLOPs needed by new prompt in a follow-up conversation request. Can be estimated as $C_1 = rC_0$, where r is the avg. ratio of the new prompt (e.g., 10%).
T_{slo}	SLO of prefill (e.g., 0.4s).
T_{load}	Avg. time to load KV cache from memory (e.g., 0.1s) using measurements from §3.1.
P	Computation power (FLOP/s) of the GPU.
P_0	FLOP/s needed for the initial request. $P_0 = C_0/T_{slo}$
P_1	FLOP/s needed for the new prompt. $P_1 = C_1/(T_{slo} - T_{load})$
R_{gpu}	Request per second (RPS) a single GPU can support. $R_{gpu} = P/(P_0(1-h) + P_1 h)$, where h is the ratio of multi-round requests.
N_{cxl}	Number of GPUs needed using our CXL memory scheme. $N_{cxl} = \lceil R/R_{gpu} \rceil = \lceil \frac{R}{P/(P_0(1-h)+P_1 h)} \rceil$
$N_{baseline}$	Number of GPUs needed without any KV cache stored (i.e., all data discarded after prefill). $N_{baseline} = \lceil \frac{R}{P/P_0} \rceil$