THEME ARTICLE: CACHE COHERENT INTERCONNECTS AND RESOURCE DISAGGREGATION TECHNIQUES

Compute-Enabled CXL Memory Expansion for Efficient Retrieval Augmented Generation

Derrick Quinn, *Cornell University, Ithaca, NY, 14850, USA*

Neel Patel, *Cornell University, Ithaca, NY, 14850, USA*

Mohammad Alian, *Cornell University, Ithaca, NY, 14850, USA*

Abstract—Conventional near-memory processing architectures often strike a trade-off between memory capacity and memory bandwidth, leading to high initial data movement or high capital costs due to memory stranding. In this work, we introduce compute-enabled memory expansion enabled by CXL as a solution for the widespread adoption of near-memory processing at scale. We present the Intelligent Knowledge Store (IKS), which is fundamentally a memory expander with lightweight near-memory accelerators that leverage high internal memory bandwidth to accelerate dense retrieval, a key component of retrieval-augmented generation (RAG). IKS disaggregates its internal memory capacity and supports both spatial and temporal multi-tenancy. It significantly accelerates high-quality dense retrieval while enabling multi-tenancy with modest memory access interference.

Near-memory processing (NMP) reduces data movement overhead by placing computation close to memory. Early products have shown promise, but there is no consensus on how to balance trade-offs in capacity, bandwidth, and complexity. Existing NMP systems often sacrifice memory capacity to achieve higher bandwidth, causing large portions of DRAM to remain underutilized and "stranded" when operating in accelerator mode. Given the capital cost and environmental impact of DRAM manufacturing, it is essential to efficiently share and utilize memory resources.

Memory disaggregation, where multiple components share pooled memory, can address stranding. However, current NMP products cannot flexibly share their internal DRAM between accelerators and CPUs. The emergence of Compute Express Link (CXL) provides a solution by enabling coherent, flexible memory expanders that can integrate computation.

We introduce the *Intelligent Knowledge Store* (IKS), a compute-enabled CXL memory expander that accelerates dense retrieval tasks, crucial in Retrieval-Augmented Generation (RAG) pipelines. Unlike traditional architectures, IKS shares its physical address space with the host CPU, allowing unused internal DRAM to serve other applications and effectively eliminating memory stranding. As illustrated in Fig. 1, IKS extends the NMP design space, balancing high internal bandwidth for specialized tasks with efficient memory sharing. More specifically, as shown in Fig. 2, IKS implements eight LPDDR5X packages, directly connected to and controlled by eight near-memory accelerator (NMA) chips, providing a total of 512 GB of DRAM capacity and an aggregate internal

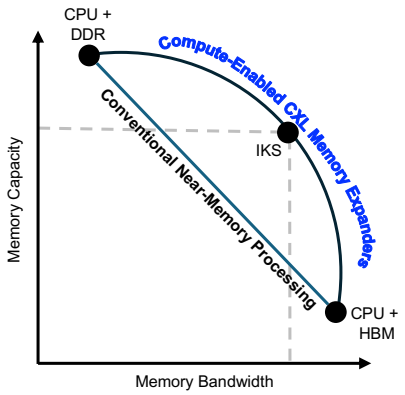


Figure 1. Spectrum of near-memory processing.

Digital Object Identifier 10.1109/XXX.0000.0000000

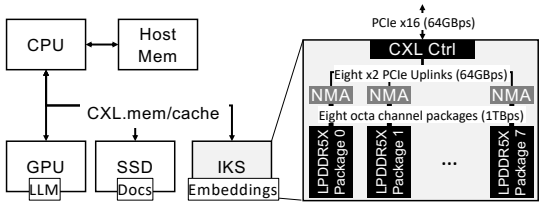
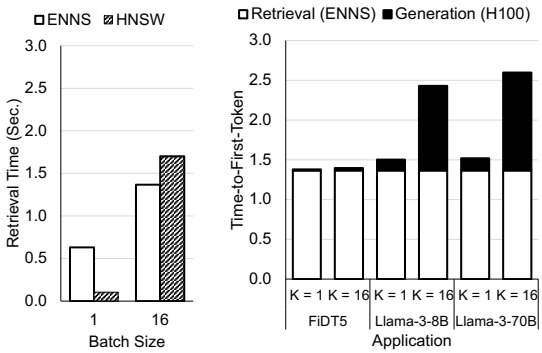


Figure 2. A server enhanced with IKS for RAG applications. IKS leverages internal bandwidth and memory parallelism of a CXL memory expander to accelerate dense retrieval.



(a) ENNS vs. HNSW **(b)** TTFT Breakdown

Figure 3. Comparison of ENNS and HNSW (Recall@32=0.95) retrieval, and Time-to-First-Token (TTFT) Breakdown for three representative RAG applications based on *Wikipedia* corpus. For generation, Llama-3-70B uses 8x NVIDIA H100 GPUs, while FiDT5 and Llama-3-8B use 1.

bandwidth of 1 TBps. The NMAs function either as memory controllers for host CPU accesses or as search engines over the embedding vectors in the LPDDR5X packages. Leveraging both `CXL.cache` and `CXL.mem`, IKS offloads computation seamlessly and achieves higher performance without complex programming overhead[9].

Our contributions are:

- We motivate the broad deployment of exact nearest neighbor search (ENNS) for dense retrieval in RAG systems.
- We propose IKS: a CXL-based memory expander that cost-effectively accelerates ENNS while minimizing memory stranding.
- We explore CPU-NMP interfaces and CXL protocols to tightly integrate near-memory acceleration into datacenter systems.
- We show that IKS effectively disaggregates its internal memory with minimal performance interference.
- We show that accelerating ENNS with IKS achieves up to 37.0× higher throughput for representative RAG applications compared to a CPU baseline using approximate nearest neighbor search.

Retrieval in Future AI Systems

Modern AI services increasingly integrate Large Language Models (LLMs) with retrieval systems, enabling Retrieval-Augmented Generation (RAG) to provide timely, contextually relevant data. Underpinning these systems are dense retrieval methods that map documents into high-dimensional vectors. Several vector search strategies exist, including Exact Nearest Neighbor Search (ENNS) as well as Approximate Nearest Neighbor Search (ANNS) methods.

While graph-based ANNS methods like HNSW can significantly limit the search space, they often suffer from complex index management, and their searches can be expensive at high accuracy targets. Although clustering-based ANNS methods like IVF have simpler indexes, they often suffer from low retrieval accuracy. In line with recent works[10], we observe that a RAG system with low retrieval accuracy can lead to an increase in the number of items (i.e., context) that need to be retrieved and sent to the downstream LLM for high-quality generation. Such an increase in context size can significantly increase the generation time and negate the end-to-end benefits of using a faster ANNS compared to a slower but exact ENNS. Moreover, in RAG serving systems in datacenters, where batching is common due to many concurrent users sending inference requests, ANNS can lose its competitive advantage compared to ENNS. As shown in Fig. 3a, when multiple queries are processed together, ENNS can reuse corpus vectors for every query in the batch, enabling it to bridge the efficiency gap with a high-quality HNSW.

Using ENNS however for RAG in current systems can significantly increase the Time-to-First-Token (TTFT), the latency before producing the initial output token. As shown in Fig. 3b, ENNS retrieval often dominates TTFT.

Combining these observations, there is a strong incentive to accelerate high-quality retrieval. ENNS's predictable, brute-force search pattern avoids the irregular data accesses and complex indexing structures of graph-based ANNS, making it inherently easier to implement and optimize in hardware. As a result, accelerating ENNS can deliver significant gains in throughput and responsiveness for next-generation RAG pipelines through simple and flexible hardware. Nevertheless, both graph-based and clustering-based ANNS algo-

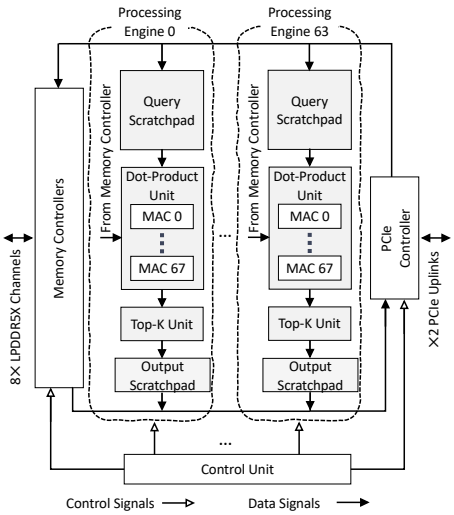


Figure 4. Architecture of a single Near-Memory Accelerator (NMA) chip. IKS implements eight NMA chips, each next to an LPDDR5X package.

rithms can benefit from an ENNS accelerator because similarity score evaluation is a core kernel shared between ENNS and all ANNS algorithms.

Intelligent Knowledge Store to Accelerate ENNS

Unlike a monolithic accelerator, IKS implements eight lightweight NMAs, each paired with an LPDDR5X package. This *scale-out* approach reduces off-chip and on-chip data movement by using smaller NMA chips instead of a large monolithic chip that can provision enough chip shoreline to connect to eight LPDDR5X packages with 64 LPDDR channels. Each LPDDR5X package provides 64 GB of DRAM capacity and eight 16-bit channels, delivering a total of 136 GBps of memory bandwidth. In contrast, achieving 512 GB of capacity using high-capacity $\times 4$ DDR5 devices would require 32 devices and a large CXL device form factor while yielding only 89.6 GBps of internal memory bandwidth. The high internal memory bandwidth is essential for IKS's near-memory acceleration of ENNS.

Near-Memory Accelerator Architecture

Within each NMA, 64 processing engines compute similarity scores between a query vector and batches of embedding vectors. A column-major data layout maps each dimension of 68 embedding vectors contiguously, allowing efficient, output-stationary dot product operations. This arrangement simplifies data distribution to MAC units and leverages batching by reusing corpus data, improving throughput and energy efficiency.

Every *vector dimension* number of clock cycles, each processing engine evaluates 68 similarity scores that need to be inserted into an ordered list maintained in the top-K unit. The insertion is overlapped with the similarity score evaluation of the next 68 vectors during the next *vector dimension* clock cycles. After all the embedding vectors are fetched from memory and evaluated for similarity, the top-K unit copies the partial top-K list into the output scratchpad. The output scratchpad is what the CPU reads and aggregates (across 8 NMAs and possibly across multiple IKS units in case of a multi-IKS setup) to construct the final top-K list.

Offload Model

The IKS address space is shared with the host CPU. The CPU runs the vector database application, which offloads the similarity calculations (i.e., dot-products between the query vectors and embedding vectors) using `iks_search(query)`, an API that does not require a system call or context switch.

When `iks_search(query)` is called, the CPU initiates a search by passing an *offload context* to IKS. As we discuss in the next section, IKS leverages `CXL.mem` and `CXL.cache` to enable CPU to seamlessly initiate offload and receive offload completion notifications with minimal overhead. The NMAs perform similarity calculations locally, returning top-K candidates that the CPU aggregates.

The CXL-Enabled Interface

This section outlines current challenges in near-memory architectures and describes how CXL-based solutions like IKS address them. We then detail the CPU-IKS interface enabled by CXL protocols.

Challenges in Current Near-Data Processing Architectures

Cost: Traditional near-memory systems often require custom hardware and software modifications, increasing complexity and deployment costs.

Stranded Resources: NMAs and their dedicated DRAM can remain underutilized, wasting capacity and increasing the total cost of ownership.

Offload Tax: Fine-grain offloads are costly. Initiating and completing offloads involves multiple PCIe round trips, diminishing the potential performance gains.

Cache Coherency: Relying on software to flush CPU cache contents to memory before initiating an offload increases offload tax. This undermines the advantage

of moving computation closer to data instead of transferring data closer to compute resources.

Address Translation: Similar to cache coherency, handling virtual addresses in accelerators can be expensive. Without an efficient translation scheme, page faults and translation lookaside buffer (TLB) misses become significant overheads.

Data Movement: Existing near-memory processing products often require explicit data transfers into accelerator-attached memory, incurring additional offload tax[7;3].

Opportunity with CXL

IKS leverages CXL features and makes several design decisions to address the challenges listed above.

Reducing Cost: IKS uses industry-standard PCIe/CXL interface for interoperability with different devices and CPUs. In addition, IKS leverages a scale-out NMA design to lower hardware complexity and improve NMA chip manufacturing yield. Mapping IKS memory directly into the host address space allows the CPU to manage vector databases, minimizing expensive code changes. To this end, we ported Meta's FAISS to IKS by adding an `IKSIndexFlatIP` library similar to the existing `GpuIndexFlatIP` used for GPU offload.

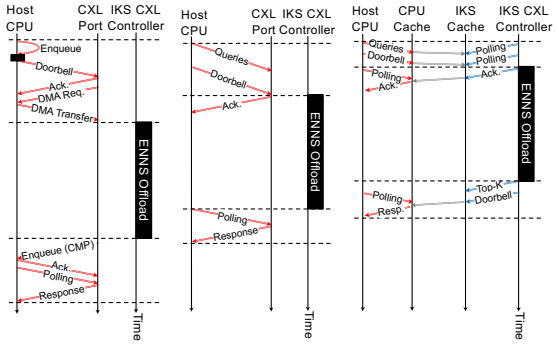
Reducing Resource Stranding: IKS disaggregates its internal DRAM, allowing it to serve as both an accelerator and a memory expander. This prevents unused capacity from going to waste and reduces overall cost. The NMAs are lightweight, minimizing idle overhead.

Reducing Offload Tax: IKS utilizes the CXL protocol to implement a low-latency interface between the CPU and NMAs, reducing the offload tax. We provide details of this interface later in this section.

Cache Coherency: IKS leverages the fact that embedding vectors are rarely updated and implements a software-managed coherency protocol where the CPU flushes the caches after the rare updates. IKS utilizes the `CXL.cache` protocol only to keep NMA control registers and scratchpads coherent. These structures total less than 200 KB of storage, maintaining a low overhead for hardware cache coherency.

Virtual Memory Translation: With `CXL.mem`, the CPU accesses IKS memory using host physical addresses. The CXL controller translates these into device media addresses for NMAs[2], which operate on a direct-segment model without page faults[1]. In IKS memory, the embedding vectors reside contiguously, simplifying address translation.

Minimizing Data Movement: CXL enables IKS to share the address space between NMA and the host CPU. Thus, NMAs directly access data in place without any data movement required before an offload .



(a) Ring Buffer **(b)** CXL.mem **(c)** CXL.cache

Figure 5. Options for IKS-CPU interface. `PIO` configuration is similar to `CXL.mem`.

IKS-CPU Interface

In this section, we explore the design space of the CPU-IKS interface and discuss the conventional interfaces (`Ring Buffer` and `PIO`) as well as those enabled by CXL. Fig. 5 illustrates and compares these interfaces.

Ring Buffer: Using a descriptor ring buffer that holds pointers to buffers allocated in the host memory and a DMA engine, an offload can be initialized by the CPU preparing a context buffer and a descriptor that points to the context buffer, pushing the descriptor to the head of the descriptor ring, and writing into a doorbell register on the IKS. Then, IKS utilizes the DMA to read the descriptor to access the pointer of the context buffer and issues another DMA access to read the context buffer from the host memory to start the offload. Once the offload is completed, IKS uses a DMA access to write the result to an output buffer and pushes a completion descriptor to the ring buffer, notifying the CPU of the completion of the offload. The CPU should either poll the completion ring or rely on interrupts from IKS to receive the notification of the offload completion.

The descriptor ring implementation has the advantage of being general, allowing multiple concurrent offloads and batching communication between the CPU and IKS for high throughput. However, this implementation suffers from high latency due to several PCIe round trips and is not suitable for fine-grain offloads.

Programmed I/O (PIO): The host CPU can use the PCIe protocol (i.e., `CXL.io`) to directly write into Memory-Mapped IO (MMIO) registers and scratchpad memory within the IKS without going through one level of indirection, which involves the ring buffer and then

DMA. In this implementation, the host CPU directly writes the context buffer to the NMAs within each IKS unit using the CXL.io protocol and then writes into a doorbell register to start the offload.

Although PIO removes the need for two DMA transactions to initiate the offload and can directly write into the MMIO registers and scratchpads of NMAs, the PIO bandwidth through `CXL.io` is limited. Such high-overhead direct writes can increase the offload initialization time for large contexts, even beyond the descriptor ring implementation.

CXL.mem Interface: The `CXL.mem` protocol enables IKS to use `CXL.mem` instead of `CXL.io` to access the MMIO registers and context scratchpads, providing higher bandwidth and lower latency compared to `CXL.io` due to cacheline flit-based multiplexing at the PCIe physical layer, compared to PCIe packet-based multiplexing at the PCIe TLP layer for `CXL.io`[2].

Although this implementation can potentially deliver much lower offload initialization latency compared to the PIO configuration, it requires CPU cycles to write the long-latency uncacheable non-temporal writes into the MMIO register and scratchpads or use temporal writes followed by a cache flush. For completion notification, this implementation needs to either rely on interrupts or polling. In the interrupt case, once an interrupt is received, the CPU needs to execute load instructions to read the result into the CPU. These loads are directly satisfied by IKS. In the polling case, the CPU should poll an uncacheable IKS-side completion register, which, due to the high latency of the loads, can take a significant number of CPU cycles. Once a completion is detected, the CPU should load the result from IKS.

CXL.cache Interface: This interface complements the `CXL.mem` interface by having the MMIO registers and scratchpad spaces coherent through `CXL.cache`, relying on a hardware-provided cache coherency protocol to communicate context buffers and notifications. This implementation not only simplifies the interface between the CPU and IKS but also reduces the overhead of initiating an offload, receiving notifications, and transferring the result to the CPU.

Disaggregating IKS Memory

IKS provides 512 GB of memory accessible to the host over a $\times$16 PCIe Gen5 link, capable of up to 64 GBps to the CPU. Internally, IKS interconnects eight LPDDR5X packages, each offering 136 GBps, for a total internal bandwidth exceeding 1 TBps – over 16$\times$ higher than the external bandwidth. This significant difference between the external and internal available memory bandwidth in IKS creates an opportunity to co-run ENNS and CPU applications with minimal interference. The memory controller inside the NMAs implements a physical address mapping hash function that maps contiguous 4 KB OS pages to a single memory channel. In this way, each OS page is colored with a channel ID. IKS exposes the page color to the OS, enabling the memory allocator to map different applications to different channels[6]. This memory allocation effectively controls where the future accesses of applications/tenants will be physically routed and enables IKS to support two modes for multi-tenancy:

Spatial Multi-Tenancy: IKS provides flexibility for fine-grain LPDDR channel partitioning to ensure quality of service and effectively eliminate interference between different classes of applications. Each LPDDR5X package implements 8 channels, each serving 8 GB of DRAM space at a theoretical bandwidth of 17 GBps. With channel partitioning, CPU and ENNS traffic can be completely isolated (because ENNS lightly utilize the external bandwidth) by mapping them to different channels.

Temporal Multi-Tenancy: Alternatively, IKS can share channels between ENNS and CPU applications. By allowing both applications to dynamically access all channels, IKS can fully leverage its internal bandwidth for ENNS when needed, maximizing ENNS performance. In this scenario, the host and ENNS may contend for bandwidth, but overall throughput and resource utilization are improved.

Methodology

We emulate IKS on a dual-socket Intel Xeon Gold 6554S server. Each socket has 512 GB of DRAM, and we use one socket to model the IKS device and NMAs, and the other as the host CPU. Although we do not use a real CXL device for our evaluation, we hypothesize that a well-optimized CXL implementation would exhibit a performance profile similar to that of a multi-socket CPU. This methodology is widely used in prior works for evaluating the performance of CXL-based systems[8].

Interfaces: We implement the `Ring Buffer`, `CXL.mem`, and `CXL.cache` interfaces described earlier. For `Ring Buffer`, we leverage the on-chip Data Streaming Accelerator (DSA) on the remote socket to emulate DMA transfers, descriptor handling, and completion notifications. This approach provides optimistic performance due to UPI latency/bandwidth differences compared to PCIe.

For `CXL.mem` and `CXL.cache`, we allocate buffers on the remote socket to emulate query and output

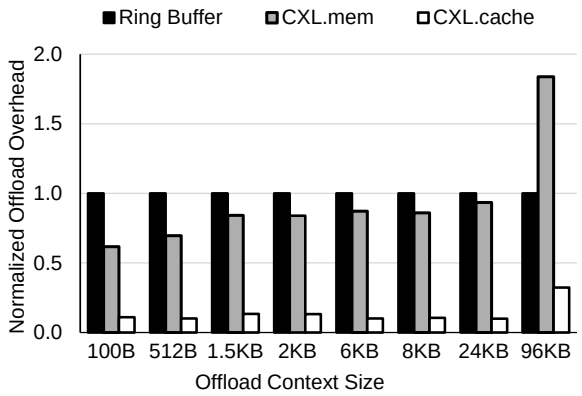


Figure 6. Comparing the offload tax (communicating context buffers) for Ring Buffer, CXL.mem, and CXL.cache

scratchpads. We use instructions like `MOVDIR64B` for direct writes and `CLDEMOTE` to model coherent cache-line operations. Completion notifications are emulated via remote writes, combined with polling (`CXL.mem`) or cache invalidations (`CXL.cache`).

Multi-Tenancy: To study spatial and temporal multi-tenancy, we use Sub-NUMA Clustering (SNC) to partition channels and emulate different memory bandwidths. Each DDR5-4000 channel stands in for two LPDDR5X channels, matching the ratio of bandwidths and allowing fair comparisons. We throttle the local CPU application's memory bandwidth to not exceed the modeled 8 GBps uplink between an NMA and CPU, aligning with the IKS design.

RAG Pipeline Evaluation: We follow Karpukhin et al.[5] to construct a Wikipedia-based corpus and fine-tune query embedding model. To evaluate generative models, we use the open-weight 8- and 70-billion parameter and Llama-3 models. Additionally, we follow Izacard and Grave[4] to fine-tune a T5-based generative model based on Google's NQ "train" dataset. From these, we construct three RAG pipelines, labeled Llama-3-8B, Llama-3-70B, and FiDT5. Queries and ground-truth answers are taken from the the NQ "dev" dataset. Each query is processed by generating a query embedding, performing retrieval (ENNS or HNSW), and then feeding top-K documents to the generative model. Accuracy for FiDT5 is exact match, while for Llama-3 models we use Rouge-L Recall.

For HNSW, we use FAISS with parameters chosen to maximize performance while maintaining a reasonable graph. The HNSW index is built with $M = 32$, `efConstruction` = 128, and runtime `efSearch` values chosen to balance speed and accuracy.

Evaluation

We evaluate the IKS-CPU interfaces, multi-tenancy (MT) modes, and overall IKS performance on RAG applications.

IKS-CPU Interface

Fig. 6 shows offload overhead for `Ring Buffer`, `CXL.mem`, and `CXL.cache`. At small context sizes, `CXL.cache` and `CXL.mem` outperform `Ring Buffer` due to reduced initialization overhead. For large contexts (e.g., 96 KB), `Ring Buffer` can leverage DMA for bulk transfers, surpassing `CXL.mem`. `CXL.cache` consistently achieves the lowest overhead, improving offload efficiency by up to 85.5% and 86.2% over `CXL.mem` and `Ring Buffer`, respectively.

IKS Multi-Tenancy and Memory Disaggregation

App	Metric	Solo Run	Spatial MT
ENNS	Search Time	313 ms	612 ms
	QPS	51.0 q/s	26.1 q/s
Memcached	P50	111µs	111 µs
	P99	191µs	199 µs
	RPS	421.9k	421.8k

TABLE 1. Comparing Memcached and ENNS performance when running solo on IKS and with spatial multi-tenancy where half of the LPDDR channels are allocated to each application. ENNS corpus size is 256 GB.

Spatial Multi-Tenancy: Table 1 compares solo and co-run configurations of ENNS and Memcached. Allocating half the channels to ENNS nearly halves its QPS, directly reflecting reduced memory bandwidth. Memcached latency and throughput remain essentially unaffected, demonstrating robust isolation of CPU traffic from ENNS.

Temporal Multi-Tenancy: Fig. 7 illustrates ENNS execution time and Memcached latency under temporal multi-tenancy. As corpus size grows, ENNS overhead rises modestly (by about 11%), and Memcached sees a small median latency increase but a more noticeable P99 increase. Temporal sharing ensures flexibility and high overall utilization, though it introduces some interference.

IKS Performance on RAG Applications

Fig. 8 compares ENNS on CPU, HNSW on CPU, and ENNS accelerated by IKS. While HNSW offers higher throughput than CPU-based ENNS, it sacrifices accuracy. With IKS acceleration, ENNS retrieval ceases to be a bottleneck, boosting throughput without

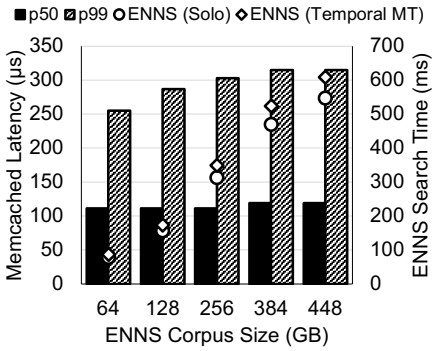


Figure 7. Latency of Memcached application and ENNS running on IKS with temporal memory disaggregation. A batch size of 16 is used. To show the impact of the impact of ENNS on Memcached, ENNS QPS is fixed to 32 with varying corpus sizes, affecting internal memory bandwidth utilization for ENNS configs. Solo is when Memcached is not co-running.

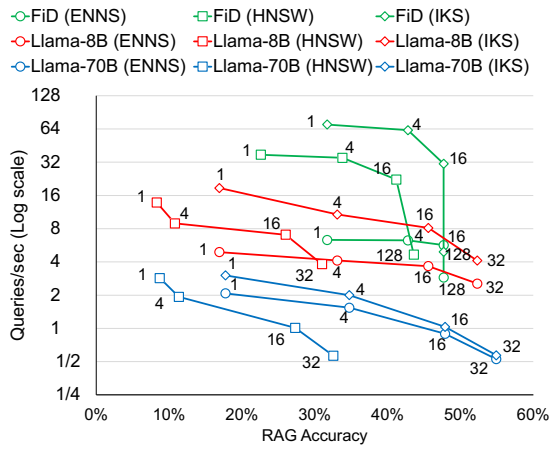


Figure 8. Comparison of accuracy and throughput of three RAG pipelines with different LLMs (FiDT5, Llama-3-8B, Llama-3-70B) with three different retrieval configurations: *ENNS* running on CPU, *HNSW* running on CPU and *IKS* which offloads ENNS to IKS. The data labels represent document count (K value). HNSW is configured with *M*, *efConstruction*, and *efSearch* of 32, 128, and 4096, respectively. Batch size 16.

diminishing accuracy. This enables high-quality RAG pipelines to achieve both high performance and accuracy, outperforming HNSW configurations.

Conclusion

In this work, we presented Intelligent Knowledge Store (IKS), a compute-enabled CXL memory expander that operates as a vector database accelerator while simultaneously disaggregating its internal memory to provide higher memory capacity. IKS leverages `CXL.mem` and `CXL.cache` to implement an optimized CPU-Accelerator interface. IKS significantly improves the performance of RAG applications by offering accelerated, scalable, and accurate dense retrieval.

Near-memory processing enabled by CXL presents numerous opportunities for further research. We see IKS as an example of a near-memory accelerator design that can leverage economies of scale to be successfully deployed, powering future compound AI systems.

REFERENCES

1. Arkaprava Basu, Jayneel Gandhi, Jichuan Chang, Mark D. Hill, and Michael M. Swift. Efficient virtual memory for big memory servers. In *Proceedings of the 40th Annual International Symposium on Computer Architecture*, ISCA '13, page 237–248, New York, NY, USA, 2013. Association for Computing Machinery.
2. Debendra Das Sharma, Robert Blankenship, and Daniel Berger. An introduction to the compute express link (cxl) interconnect. *ACM Comput. Surv.*, 56(11), July 2024.
3. Fabrice Devaux. The true processing in memory accelerator. In *2019 IEEE Hot Chips 31 Symposium (HCS)*, pages 1–24, 2019.
4. Gautier Izacard and Edouard Grave. Leveraging passage retrieval with generative models for open domain question answering. In Paola Merlo, Jorg Tiedemann, and Reut Tsarfaty, editors, *Proceedings of the 16th Conference of the European Chapter of the Association for Computational Linguistics: Main Volume*, pages 874–880, Online, April 2021. Association for Computational Linguistics.
5. Vladimir Karpukhin, Barlas Oğuz, Sewon Min, Patrick Lewis, Ledell Wu, Sergey Edunov, Danqi Chen, and Wen tau Yih. Dense passage retrieval for open-domain question answering. *arXiv*, cs.CL, 2020.
6. R. E. Kessler and Mark D. Hill. Page placement algorithms for large real-indexed caches. *ACM Trans. Comput. Syst.*, 10(4):338–359, nov 1992.
7. Sukhan Lee, Shin-haeng Kang, Jaehoon Lee, Hyeonsu Kim, Eojin Lee, Seungwoo Seo, Hosang Yoon, Seungwon Lee, Kyounghwan Lim, Hyunsung Shin, Jinhyun Kim, O Seongil, Anand Iyer, David Wang, Kyomin Sohn, and Nam Sung Kim. Hardware architecture and software stack for pim based on commercial dram technology : Industrial product. In *2021 ACM/IEEE 48th Annual International Symposium on Computer Architecture (ISCA)*, pages 43–56, 2021.
8. Huaicheng Li, Daniel S. Berger, Lisa Hsu, Daniel

Ernst, Pantea Zardoshti, Stanko Novakovic, Monish Shah, Samir Rajadnya, Scott Lee, Ishwar Agarwal, Mark D. Hill, Marcus Fontoura, and Ricardo Bianchini. 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*, ASPLOS 2023, page 574–587, New York, NY, USA, 2023. Association for Computing Machinery.

9. Derrick Quinn, Mohammad Nouri, Neel Patel, John Salihu, Alireza Salemi, Sukhan Lee, Hamed Zamani, and Mohammad Alian. Accelerating retrieval-augmented generation. In *Proceedings of the 30th ACM International Conference on Architectural Support for Programming Languages and Operating Systems, Volume 1*, ASPLOS '25, page 15–32, New York, NY, USA, 2025. Association for Computing Machinery.
10. Shi-Qi Yan, Jia-Chen Gu, Yun Zhu, and Zhen-Hua Ling. Corrective retrieval augmented generation. *arXiv*, 2024. arXiv:2401.15884.

Derrick Quinn is a Ph.D. student in Electrical and Computer Engineering at Cornell University. Derrick's research interests are computer architecture and systems at scale. Contact him at dq55@cornell.edu.

Neel Patel is a Ph.D. student in Electrical and Computer Engineering at Cornell University. Neel's research interests are computer architecture and systems at scale. Contact him at nmp83@cornell.edu.

Mohammad Alian is an Assistant Professor in Electrical and Computer Engineering at Cornell University. He completed his Ph.D. at the University of Illinois Urbana Champaign. His research team is focused on redefining the data-delivery hierarchy of future data centers. Contact him at malian@cornell.edu.