Accelerating Retrieval-Augmented Generation

Derrick Quinn
Cornell University
dq55@cornell.edu

Mohammad Nouri
Cornell University
mn636@cornell.edu

Neel Patel
Cornell University
nmp83@cornell.edu

John Salihu
University of Kansas
jsalihu@ku.edu

Alireza Salemi
University of Massachusetts Amherst
asalemi@cs.umass.edu

Sukhan Lee
Samsung Electronics
sh1026.lee@samsung.com

Hamed Zamani
University of Massachusetts Amherst
zamani@cs.umass.edu

Mohammad Alian
Cornell University
malian@cornell.edu

Abstract

An evolving solution to address hallucination and enhance accuracy in large language models (LLMs) is Retrieval-Augmented Generation (RAG), which involves augmenting LLMs with information retrieved from an external knowledge source, such as the web. This paper profiles several RAG execution pipelines and demystifies the complex interplay between their retrieval and generation phases. We demonstrate that while exact retrieval schemes are expensive, they can reduce inference time compared to approximate retrieval variants because an exact retrieval model can send a smaller but more accurate list of documents to the generative model while maintaining the same end-to-end accuracy. This observation motivates the acceleration of the exact nearest neighbor search for RAG.

In this work, we design Intelligent Knowledge Store (IKS), a type-2 CXL device that implements a scale-out near-memory acceleration architecture with a novel cache-coherent interface between the host CPU and near-memory accelerators. IKS offers 18–52× faster exact nearest neighbor search over a 512GB vector database compared with executing the search on Intel Sapphire Rapids on-chip accelerators. This higher search performance translates to 2.0–49× lower end-to-end inference time for representative RAG applications. IKS is inherently a memory expander; its internal DRAM can be disaggregated and used for other applications running on the server to prevent DRAM – which is the most expensive component in today's servers – from being stranded.

1 Introduction

State-of-the-art natural language processing systems heavily rely on large language models (LLMs)–deep Transformer networks [95] with hundreds of millions of parameters. There is much evidence that information presented in the LLM training corpora is "memorized" in the LLM parameters, forming a parametric knowledge base that the model depends on for generating responses. A major challenge with parametric knowledge is its static nature; it cannot be updated unless the model undergoes retraining or fine-tuning, which is an extremely costly process. This creates a critical issue, especially when it comes to non-stationary domains where fresh content is constantly being produced [108]. Besides, previous studies have indicated that LLMs exhibit limited memorization for less frequent entities [33], are susceptible to hallucinations [85], and may experience temporal degradation [35].

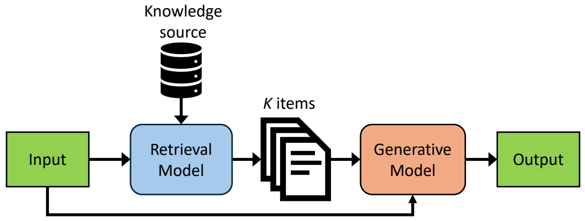


Fig. 1. Overview of the Retrieval-Augmented Generation (RAG) pipeline.

To overcome the challenges presented by LLMs, a potential solution is to enhance them with non-parametric knowledge, where the LLM is augmented with information retrieved from a knowledge source (e.g., text documents). These approaches have recently gained considerable attention in the machine learning communities [26, 29, 45, 53, 62, 80, 85], and have played key roles in some recent breakthrough applications in the tech industry, such as Google Gemini [92], Microsoft Copilot [88], and OpenAI ChatGPT with Retrieval Plugins [60]. Retrieval-Augmented Generation (RAG) is the term that is used to refer to systems that adopt this approach in the context of LLMs.

A RAG application includes two key components: a retrieval model and an LLM for text generation, called the generative model. When a query is received, the retrieval model searches for relevant items (e.g., documents) and the top retrieved items, together with the input, are sent to the generative model. Current state-of-the-art retrieval approaches use bi-encoder neural networks (called dense retrieval) [34] for learning optimal embedding for queries and documents. Each item is then encoded into a high-dimension vector (called embedding vectors) and stored in a vector database. Such approaches use K-nearest neighbor algorithms for retrieving the top "K" items from the vector database. Figure 1 provides an overview of a RAG application.

Accepted for publication at ASPLOS 2025

The accuracy of generated output in RAG hinges on the quality of the retrieved item list. Conducting an Exact Nearest Neighbors Search (ENNS) to retrieve the precise top K relevant items involves scanning all the embedding vectors in the vector database, which is costly in today's memory bandwidth-limited systems. For example, in a RAG application with a 50GB (using a 16-bit floating point representation) vector database running on an Intel Xeon 4416+ with 8×DDR5-4000 memory channels, and a generative model running on an NVIDIA H100 GPU, ENNS takes up to 97% of the end-to-end inference time (§3).

One strategy to mitigate the retrieval cost is to employ Approximate Nearest Neighbor Search (ANNS), and opt for a faster, but lower-quality search configuration. While lower-quality retrieval can improve search time, our extensive experiments on Question Answering applications demonstrate that a lower-quality search scheme should provide significantly more items to the language model in order to match the end-to-end RAG accuracy of ENNS or a higher-quality, but slower ANNS configuration. This virtually negates any benefits gained during the retrieval phase and even increases the end-to-end inference time.

In this paper, we extensively profile the execution pipeline of RAG, demystifying the complex interplay between various hardware and software configurations in RAG applications [1]. Motivated by the need for high-performance, low-cost, high-quality search and the limitations of current commodity systems, we contribute the Intelligent Knowledge Store (IKS), a cost-optimized, purpose-built CXL memory expander that functions as a high-performance, high-capacity vector database accelerator. IKS offloads memory-intensive dot-product operations in ENNS to a distributed array of low-profile accelerators placed near LPDDR5X DRAM packages.

IKS implements a novel interface atop the CXL.cache protocol to seamlessly offload exact vector database search operations to near-memory accelerators. IKS is exposed as a memory expander that disaggregates its internal DRAM capacity and shares it with vector database applications and other co-running applications through CXL.mem and CXL.cache protocols. Instead of building a full-fledged vector database accelerator, IKS co-designs the hardware and software to implement a minimalist scale-out near-memory accelerator architecture. This design relies on software to map data into the internal IKS DRAM and scratchpads while performing the final top-K aggregation.

In summary, we make the following contributions:

- We demystify RAG by profiling its execution pipeline. We explore various hardware, system, and application-level configurations to assess the performance and accuracy of RAG.
- We demonstrate that RAG requires high-quality retrieval to perform effectively; nonetheless, current RAG applications are bottlenecked by a high-quality retrieval phase.
- We introduce Intelligent Knowledge Store (IKS), which is a specialized CXL-based memory expander equipped with low-profile accelerators for vector database search. IKS leverages CXL.cache to implement a seamless and efficient interface between the CPU and near-memory accelerators.
- We implemented an end-to-end accelerated RAG application using IKS. IKS accelerates ENNS for a 512GB knowledge store by 18–52×, leading to a 2.0–49× end-to-end inference speedup for representative RAG applications.

2 Background

2.1 Information Retrieval in RAG

Recent advancements in RAG indicate superior outcomes when employing dense retrieval over other methods, for uni-modal [29, 30, 45] and multi-modal [23, 77, 78] scenarios. Consequently, our emphasis in this study centers on dense retrieval models, exploring their efficiency-related aspects.

In the context of dense retrieval, a query encoder, denoted as E_q, and a document encoder, denoted as E_d, are trained to encode queries and documents, respectively, and map them into a high-dimensional vector space. The similarity score between a document[1] d and a query q is calculated as $s_d = E_q(q) \cdot E_d(d)$, where $E_q(q) \in \mathbb{R}^h$, $E_d(d) \in \mathbb{R}^h$ and h is the hidden dimension of query and document encoders. Then documents are sorted based on their similarity scores and top documents are retrieved [34]. In a real RAG implementation, in an offline phase, all the documents are encoded into embedding vectors. The embedding vectors are stored in a vector database for dense retrieval. In the paper, we refer to the encoded documents as *embedding vectors* and the vectors generated by the retriever model as *query vectors*.

For dense retrieval, two distinct search algorithms are prevalent: Exact Nearest Neighbor Search (ENNS) and Approximate Nearest Neighbor Search (ANNS). ENNS computes the pair-wise distance matrix between embedding and query vectors, prioritizing accuracy in similarity measurement. In ANNS, however, strategies such as Product Quantization (PQ) [32], Inverted File with Product Quantization (IVFPQ) [10], and Hierarchical Navigable Small World (HNSW) [55], are employed to trade off search accuracy for higher search efficiency.

2.2 Applications of RAG

RAG has proven beneficial for various tasks in natural language processing [40, 48, 108], including dialogue response generation [6, 11, 13, 85, 93, 94, 98–100], machine translation [22, 25, 103, 109], grounded question answering [29, 30, 45, 68, 69, 73, 74, 76, 87, 107], abstractive summarization [62, 67], code generation [24, 53], paraphrase generation [36, 91], and personalization [41, 75, 79, 80]. Additionally,

[1] The term "document" refers to any retrievable item from the knowledge source.

RAG's application extends to multi-modal data tasks like caption generation from images, image generation, and visual question answering [14, 15, 19, 23, 72, 77, 82].

It is noteworthy that commercial LLM systems employing RAG are typically proprietary, and as such, their implementations are not openly accessible. Nevertheless, insights into the implementation of these systems can be gleaned from open-source releases by research labs within commercial entities. We adhere to a methodology akin to the approach outlined by Izacard and Grave [29] and Lewis et al. [45], both of which are contributions from Meta AI. Our implementations closely align with the depicted pipeline in Figure 1. Specifically, we employ a dense document retrieval model as the retriever and leverage a language model for answer generation, consistent with the aforementioned work. Additionally, for efficient vector search capabilities, we utilize the Faiss [31] library, similar to the aforementioned works.

3 Demystifying RAG

In this section, we profile the end-to-end execution of three representative long-form question-answering RAG applications and quantify both the execution time breakdown and the generation accuracy of RAG with different hardware and software configurations: FiDT5, where we use the T5-based Fusion-in-Decoder [29, 70] as the generative model, as well as L3-8B, and L3-70B, where we use 4-bit-GGUF-Quantized Llama-3-8B-Instruct and Llama-3-70B-Instruct [56] as the generative models, respectively. The knowledge source for all workloads is Wikipedia, and a trained BERT base (uncased) model is used to generate embedding vectors for documents. We test with 50GB and 512GB (assuming 16-bit number format) vector database sizes (corpus size) that store the embedding vectors. The documents themselves are stored in the CPU memory.

In FiDT5, the documents are presented via the Fusion-in-Decoder approach, where documents are encoded by the encoder stage of a T5 model, and these encoded representations are combined for the decoder stage. In L3-8B and L3-70B, retrieved documents are presented as plaintext in the prompt. For more information about the methodology, see Section 6.1.

In the following sections, we discuss both the accuracy of an end-to-end RAG system and the retrieval model on its own. *Retrieval accuracy* is discussed in terms of recall, where ENNS is considered to be perfect, and the recall score of an ANNS algorithm is the proportion of relevant documents retrieved by both ENNS and ANNS algorithm compared to the total number of relevant documents retrieved by ENNS. *Generation accuracy* refers to how well an end-to-end RAG system answers questions. For details on the evaluation of generation accuracy, see Section 6.2.

3.1 Tuning RAG Software Parameters

Both the retrieval phase and generation phase of the RAG models that we use offer support for batching of queries in order to improve data reuse and to amortize data movement overheads over several queries. However, batching is not always an option in practice, particularly in the case of latency-critical applications. We consider batch sizes of 1 for latency-critical uses across all applications, and 16 for throughput-optimized applications. Batch size does not impact generation accuracy and only affects execution time. An important parameter in RAG is "K" or the *number of documents* retrieved and fed to the language model for text generation. Increasing the document count significantly impacts the generation time. The computation required for transformer inference scales at least linearly with the input size [95], and if we concatenate the retrieved documents, we face significant computation and memory overhead [17, 112]. In particular, the memory required to store a key-value cache entry for a single token can be computed as follows: $n_{\text{layers}} \times n_{\text{KV-heads}} \times d_{\text{head}} \times n_{\text{bytes}} \times 2$, where n_{bytes} refers to the size of the number format [50]. For L3-8B with a 16-bit number format, this is $32 \times 8 \times 128 \times 2 \times 2 = 131$ kB per token. While exact token counts depend on the tokenization process, each document (for all applications) is 100 words long; for L3-8B and L3-70B, this averaged 127 tokens per document across our evaluation dataset.

3.2 Examining Approximate Search for RAG

An important algorithmic consideration that can impact the inference time and generation accuracy of RAG is the choice of retrieval algorithm from the vector database, where we can use exact nearest neighbor search (ENNS) or approximate nearest neighbor search (ANNS). The particular algorithm used for retrieval is implemented by a data structure called an *index*, which stores the embedding vectors computed offline, as described in Section 2.1. For ENNS, an index is a wrapper around an array of embedding vectors sequentially iterated over during the search, but for ANNS, the index can be more complex. For example, HNSW stores embedding vectors in a graph-based data structure [55].

To evaluate ANNS, we use the state-of-the-art HNSW [55] ANNS algorithm, and fine-tune the *M* and *efConstruction* parameters to maximize retrieval accuracy without severely increasing runtime, yielding an index with *M* of 64 and *efConstruction* of 128. From this, we evaluate two configurations, ANNS-1 and ANNS-2, which use different *efSearch* parameters: 1024 and 10000. In the context of an end-to-end RAG system, the trade-off of generation accuracy and runtime was evaluated for this index for various choices of *efSearch*. A lower *efSearch* provides higher search throughput, but lower generation accuracy, and a higher *efSearch* provides lower search throughput, but higher generation accuracy. Other HNSW and IVFPQ indexes were tested but provided lower generation accuracy, or similar runtime to ENNS (or even worse, in some cases), negating the benefits of approximation.

Generation Accuracy with ANNS vs. ENNS: Figure 2 compares the generation accuracy and throughput of ANNS- and ENNS-based RAG applications for FiDT5, L3-8B, and L3-70B. The figure illustrates that retrieval quality strongly influences

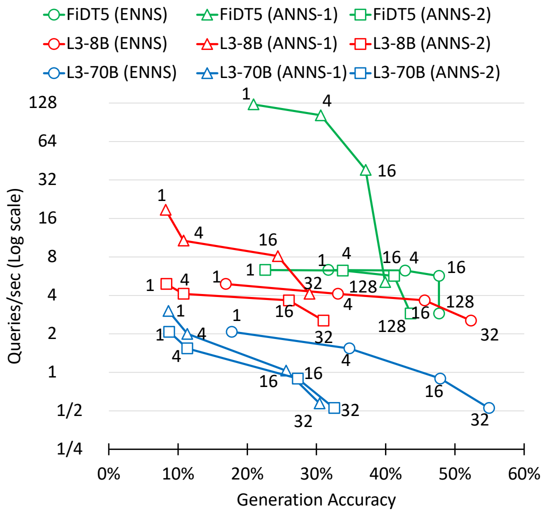
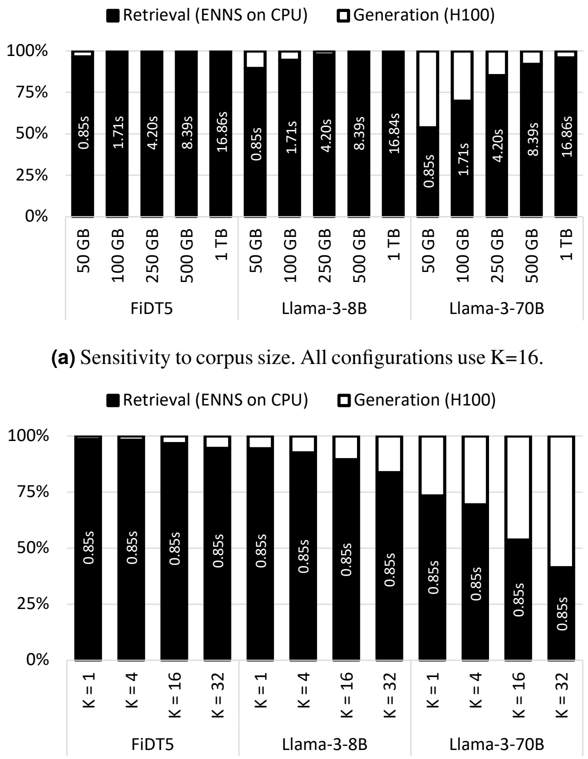


Fig. 2. Generation accuracy vs. throughput (Queries/sec) of representative RAG applications for various retrieval algorithms and document counts (K). The label on the data points is the value of K. A throughput-optimized configuration is used, with a batch size of 16 for all generative models.

the end-to-end generation accuracy. As shown in Figure 2, with document count of one, compared to ENNS, the generation accuracy of ANNS-1 and ANNS-2 drops by 22.6 and 34.0% for `FiDT5`, 52.8 and 53.4% for `L3-8B`, and 51.0 and 51.5% for `L3-70B`, respectively. With a document count of 16, a similar trend in generation accuracy is observed, with ANNS-1 and ANNS-2 leading to an accuracy reduction of 13.6 and 22% for `FiDT5`, 38.4 and 42.2% for `L3-8B`, and 38.4 and 45.2% for `L3-70B`, respectively. Interestingly, the impact of retrieval quality on generation accuracy appears to be even larger when using large models that have not been fine-tuned for this task.

Several prior works [9, 110] demonstrate that hyper-parameter tuning can enhance the retrieval accuracy of ANNS, potentially matching that of ENNS across various workloads. While we optimized our HNSW indexes for accuracy and throughput, these indexes could not match ENNS in end-to-end generation accuracy while achieving significantly (more than 2×) faster search. By using a small *efSearch* value, retrieval speed improves significantly, allowing for the use of a larger value of K to compensate for the reduced retrieval quality. However, trading retrieval quality for retrieval speed in this way resulted in lower generation accuracy and end-to-end throughput compared to a larger *efSearch*, where a higher-quality, slower search scheme permits greater accuracy at lower K values (thus reducing generation times). For example, ANNS-2 with 16 documents have 3% higher accuracy and 128% higher throughput compared to ANNS-1 with 128 documents for `FiDT5`. Further improving retrieval quality via exact search gives ENNS-based RAG Pareto-superiority above sufficiently high accuracy thresholds (~43%, ~27%, and ~14% for `FiDT5`, `L3-8B`, and `L3-70B`, respectively) as demonstrated in Figure 2. In general, our findings highlight the potential for reducing generation time by leveraging high-quality retrieval methods when high accuracy is required.



(a) Sensitivity to corpus size. All configurations use K=16.

(b) Sensitivity to K. All configurations use a 50 GB corpus.

Fig. 3. Latency breakdown of `FiDT5`, `L3-8B`, `L3-70B` for various values of K, corpus sizes. All configurations use batch size 1. Retrieval is ENNS and runs on CPU, generation runs on a single NVIDIA H100 (SXM) for all generative models. The value in each bar shows the absolute retrieval time.

Scaling of ANNS and ENNS: Previous works [9, 55] identified the trade-off between retrieval quality and runtime, and challenges with high-quality ANNS have motivated accelerators such as ANNA [44] and NDSearch [97]. While lower-quality ANNS algorithms could possibly provide orders of magnitude faster nearest neighbor search compared with ENNS, high-quality ANNS algorithms are shown to provide only a modest speedup [49, 96]. For example, ANNS-2, which is the best performing ANNS configuration in Figure 2, offers only a 2.5× speedup compared with ENNS. In fact, all the Pareto frontier configurations that provide high generation accuracy in Figure 2 are ENNS. Therefore, in the rest of this section, we focus on understanding how to optimize and accelerate RAG applications with ENNS.

3.3 End-to-End RAG Performance with ENNS

In this subsection, we profile time-to-interactive (also known as time to first token) [89] for the `FiDT5`, `L3-8B`, and `L3-70B` RAG applications and report latency ratios for the retrieval and

Batch Size	1		16	
Corpus Size	50	512	50	512
CPU	1	1	1	1
AMX	1.05	1.02	1.10	1.09
GPU	7.1	49.5	11.4	80.0

Table 1. Speedup of Intel AMX and GPU for ENNS, relative to a CPU baseline. AMX speedup is flat for very small batch sizes, due to the memory-bound nature of similarity search. For 50GB and 512GB corpus size, 1 and 7 H100 GPUs are used, respectively.

generation phases. For all experiments, retrieval uses ENNS and runs on the CPU, while generation runs on a single NVIDIA H100 GPU. We select CPU as the baseline for ENNS retrieval, rather than GPU. This decision is made based on the high cost of using GPU memory

As we discussed in Section 3.2, the generation accuracy of RAG applications directly depends on the retrieval accuracy. However, as shown in Figure 3a, utilizing ENNS for retrieval can quickly become an end-to-end bottleneck in RAG applications, even for large models. Although it is possible to compensate for the retrieval accuracy by increasing K (in case of using ANNS), as shown in Figure 3b, increasing K would increase the generation time and is costly in terms of time to first token.

The two phases in a RAG pipeline have different characteristics: ENNS is extremely memory bandwidth-bound, and generation is relatively compute-bound. Nevertheless, the current state-of-the-art focus in building AI systems is only on accelerating the generation phase [5, 8, 21, 38, 52, 57, 66, 84, 101, 102, 105]. Next, we discuss the feasibility of accelerating high-quality nearest neighbor search for future RAG applications.

3.4 High-Quality Nearest Neighbor Search Acceleration

Given the sensitivity of RAG generation accuracy, latency, and throughput to the retrieval quality, it is imperative to focus exclusively on accelerating the retrieval phase of future RAG applications. In this subsection, we discuss the feasibility of accelerating high-quality ANNS and ENNS.

Acceleration of High-Quality ANNS: High-quality ANNS can be as slow as ENNS [49]. There are prior works aimed at building hardware accelerators for high quality ANNS [44, 97] because GPUs are not effective at accelerating key ANNS algorithms such as IVFPQ and HNSW [31]. Unfortunately, the complex algorithms and memory access patterns used for ANNS algorithms also make ANNS accelerators highly task-specific; for example, ANNA [44] and NDSearch [97] can only accelerate PQ-based and graph-based ANNS algorithms, respectively. However, our experimental results, which are in line with prior findings [96], show that different corpora are amenable to different ANNS algorithms.

Acceleration of ENNS: ENNS can be accelerated using conventional SIMD processors such as GPUs and Intel AMX because the algorithm is simple and data-parallel. Table 1 compares the speedup of AMX and GPU against a CPU baseline. Although GPUs can significantly speed up ENNS, as the corpus size increases, the cost of offloading ENNS to GPUs increases significantly. For example, to fit the 50GB and 512GB corpus sizes tested in Table 1, we need to use 1 and 7 H100 GPUs, respectively. One of the key contributors to the cost of GPUs is the high-bandwidth memory (HBM) used to implement GPU main memory, which is more than 3× more expensive than DDR or LPDDR-based memories [58]. Lastly, GPUs provision huge amounts of compute relative to memory bandwidth[2], meaning that a large GPU die is poorly utilized by the primarily memory-bound workload of ENNS [28].

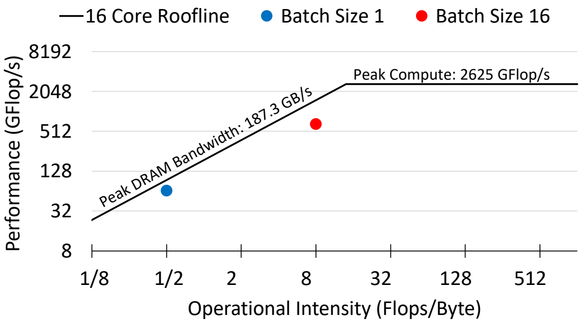


Fig. 4. Roofline model for ENNS using Batch Size 1 and 16 See Sec.6.1 for experimental setup.

3.5 Summary

The analysis presented in this section, using various software and system configurations for RAG applications, led to the following takeaways:

- Generation accuracy, time to interactive, and throughput of RAG applications can be improved by using a slower but higher-quality retrieval scheme.
- When high-quality retrieval is used, the retrieval phase accounts for a significant portion of end-to-end runtime, regardless of whether the search is performed via ENNS or high-quality ANNS.
- Using GPUs to accelerate ENNS is expensive, and GPUs are not able to accelerate high-quality ANNS effectively or affordably.
- New accelerators for ANNS are highly complex and task-specific due to the unique requirements of ANNS algorithms, while ENNS relies on a very simple scheme, making ENNS simpler to accelerate than ANNS.

4 A Case for Near-Memory ENNS Acceleration

ENNS is characterized by the following features:

- ENNS operations exhibit no data re-use for pair-wise similarity score calculations between corpus vectors and a query vector.
- ENNS operations consist of simple vector-vector dot-products coupled with top-K logic.

[2] NVIDIA H100 80GB provisions 296 Flops/Byte and 592 Int8 Ops/Byte

- ENNS has a regular and predictable memory access pattern.
- ENNS is highly parallelizable, allowing the corpus to be distributed across different processors with a simple aggregation of top-K similarities at the end.

These features make ENNS a prime candidate for near-memory acceleration due to the following reasons: (1) Deep cache hierarchies are not beneficial for ENNS and can even cause slowdown due to the complex cache maintenance and coherency operations managed by the hardware. This is evident from the roofline model in Figure 4 as ENNS running on the CPU cannot saturate the available DRAM bandwidth. (2) The limited data reuse with huge data set size enables low overhead software-managed cache coherency implementation between host CPU and near-memory accelerators. (3) The regular memory access pattern of ENNS enables coarse-grain virtual to physical address translation on near-memory accelerators. (4) ENNS operations can be efficiently offloaded to a distributed array of near-memory accelerators that each operate in parallel on a shard of corpus data with a low-overhead top-K aggregation phase at the end.

Leveraging these unique features, we design, implement, and evaluate Intelligent Knowledge Store (IKS), a memory expander with a scale-out near-memory acceleration architecture, uniquely designed to accelerate vector database search in future scalable RAG systems. IKS is designed with three requirements in mind: (1) The memory capacity of IKS should be cost-effective and scalable because the size of vector databases for RAG applications is several tens or hundreds of gigabytes and is likely to increase. (2) The near-memory accelerators should be managed in userspace as the cost of context switches and kernel overhead would reduce the benefits of offloads. (3) The near-memory accelerators and host CPU should implement a shared address space; otherwise, explicit data movements between the CPU and near-memory accelerator address spaces will negate the benefits of near-memory offloads; another issue that GPU acceleration of ENNS suffers from. Moreover, a partitioned address space requires rewriting the entire vector database application, as ENNS is just one operation we want to accelerate near the memory, while other data manipulation operations, such as updates, should be managed by the host CPU.

We designed IKS, a type-2 CXL memory expander/accelerator, to meet all these requirements. Our rationale for choosing CXL over DDR-based (or DIMM-based) [7, 39, 65, 111] near-memory processing architecture is that DIMM-based near-memory processing (1) requires sophisticated mechanisms to share the address space between near-memory accelerators and the host [64], (2) limits per-rank memory capacity and compromises the memory capacity of the host CPU when used as an accelerator, and (3) has limited compute and thermal capacity. Instead, IKS relies on asynchronous CXL.mem and CXL.cache protocols to safely share the address space and independently scale the local and far memory capacity of the host CPU, implement a low-overhead interface for offloading from the userspace, and eliminate the limitations on the compute or thermal capacity of the PCIe-attached IKS card. In Section 5, we explain the architecture of IKS and its interface to the host CPU, and how we used it to accelerate an end-to-end RAG application.

5 Intelligent Knowledge Store

5.1 Overview

Figure 5a provides an overview of the Intelligent Knowledge Store (IKS) architecture. IKS incorporates a scale-out near-memory processing architecture with low-profile accelerators positioned near the memory controllers of LPDDR5X packages. While IKS can function as a regular memory expander, it is specifically designed to accelerate ENNS over the embedding vectors stored in its LPDDR5X packages.

As shown in Figure 5a, IKS utilizes eight LPDDR5X packages, each directly connected to a Near-Memory Accelerator (NMA) that implements both an LPDDR5X memory controller and accelerator logic. Each package contains 512Gb LPDDR5X DRAM with eight 16-bit channels, similar to CXL-PNM [61] and MTIA [20]. One of the key differences between IKS and CXL-PNM and MTIA is the *scale-out* near-memory acceleration architecture. IKS distributes the NMA logic over multiple chips, each providing high-bandwidth and low-energy access to its local LPDDR5X package.

Why Scale-Out NMA Architecture? The rationale for such a scale-out NMA architecture is to keep the area of the NMA chip in check. Because memory PHYs are only implemented at the shoreline of a chip [4, 20, 54], to implement 64 LPDDR5X memory channels, we need a chip with an approximate perimeter of 160 mm. This is because each LPDDR5X channel PHY approximately occupies a shoreline of 2.5 mm, based on the die shots of Apple M2 [63] in 5nm technology. A square-shaped chip with a 160 mm perimeter has an area of 1600 mm^2, which is larger than the state-of-the-art lithography reticle limit [3]! Although we can technically manufacture such a large accelerator using chiplets, the area of this huge multi-chip module would be wasted, as it is much larger than what is needed to implement the NMA logic, memory controllers, and PCIe/CXL controllers. For context, the area of an H100 GPU is 814 mm^2.

Splitting the NMAs into smaller chips increases the aggregate chip shoreline and improves yield. Using one NMA per LPDDR5X package requires only eight LPDDR5X memory channels per NMA, necessitating a minimum chip perimeter of 20 mm. IKS implements ×4 PCIe 5.0 to provide a 16 GBps uplink connecting each NMA to the CXL controller. With this design, the uplinks to the CXL controller are oversubscribed. Nevertheless, this oversubscription is neither a bottleneck for IKS operating in acceleration mode nor for IKS operating in memory expander mode. In acceleration mode, the bandwidth of local LPDDR5X channels is utilized for dot product calculations, and in memory expander mode, the data is interleaved

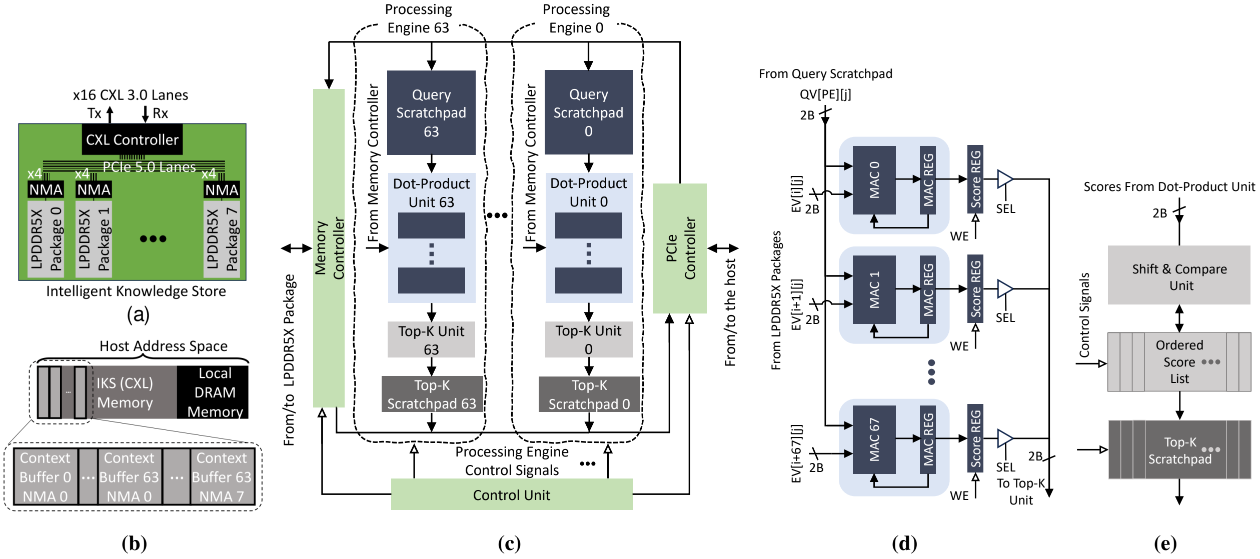


Fig. 5. (a) IKS includes eight LPDDR5X packages with one near-memory accelerator (NMA) chip near each package. (b) IKS internal DRAM, scratchpad spaces, and configuration registers are mapped to the host address space. The scratchpad and configuration register address ranges are labeled as *Context Buffers*. (c) Each NMA includes 64 processing engines. (d) Each dot-product unit includes 68 MAC units. (e) Top-K unit and Top-K scratchpad structure.

over multiple LPDDR5X packages and read in parallel over the multiple ×4 PCIe uplinks.

IKS is a type 2 CXL device. IKS's internal memory is exposed as host-managed device memory where both the CPU and IKS can cache any addresses within this unified address space (Figure 5b). IKS leverages the low-latency accesses of CXL.mem and CXL.cache protocols to implement a novel interface between the near-memory accelerators and the CPU that: (1) eliminates the need for DMA setup and buffer management, and (2) eliminates the overhead of interrupt and polling for implementing notifications between the CPU and near-memory accelerators (§5.3).

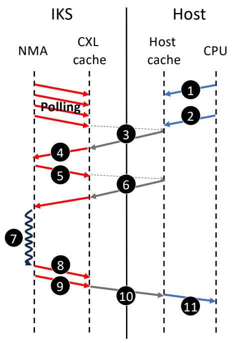
IKS supports spatial and coarse-grain temporal multi-tenancy. In spatial multi-tenancy, the IKS driver partitions embedding vectors that belong to different vector databases across different packages, allowing each NMA to execute ENNS independently per vector database. For temporal multi-tenancy, the IKS driver time-multiplexes similarity search in NMAs among different vector databases that store their embedding vectors in the same LPDDRX5 package. Time multiplexing takes place at the boundary of a complete similarity search.

Why LPDDR? For IKS, a customized type-2 CXL device that should support cost-effective high capacity, neither HBM (expensive) nor DDR (general-purpose) are good options. LPDDR DRAM packages are integrated as part of system-on-chip designs, resulting in shorter interconnections, faster clocking, and less power wastage during data transmission. The most recent release of LPDDR, LPDDR5X, offers a bandwidth of 8533 Mbps per pin, exceeding that of DDR5, which provides a bandwidth of 7200 MTps. However, one challenge with using LPDDR in a datacenter setting is reliability, as LPDDR was originally designed for mobile systems. Although we could provision an in-line ECC processing block for error detection and correction, ENNS similarity search is resilient to bit flips, and rare bit flips in ENNS have negligible impact on the end-to-end RAG accuracy.

5.2 Offload Model

The IKS address space is shared with the host CPU. The host CPU stores embedding vectors with a specific data layout (that we discuss in Section 5.5) in contiguous physical addresses in IKS, while the actual documents are stored in the host memory (either in DDR memory or CXL memory). 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)`, a blocking API that does not require a system call or context switch. After each update operation, the vector database application will flush CPU caches to ensure that when `iks_search(query)` is called, IKS does not contain any stale values.

`iks_search(query)` hides the complexity of interacting with IKS hardware from the programmer by writing an *offload context* to IKS and initiates an offload by writing into a doorbell register. The offload context and doorbells are communicated through memory-mapped regions called *context buffers* to the IKS as shown in Figure 5b. An *offload context* includes query vectors, vector dimensions, and the base address of the first embedding vector stored in each LPDDR5X package. The host process then uses `umwait()` to block on the doorbell register (shared between IKS and the host and kept coherent via the CXL.cache protocol) to implement efficient notification between the paused CPU process and near-memory accelerators [106].



	Description
1	CPU writes the parameters and queries to the corresponding NMA context buffers
2	CPU writes into the NMA context doorbell and pauses on the doorbell updates using umwait()
3	NMA polling on the doorbells leads to cache coherency activity, moving the doorbell to IKS
4	NMA gets informed of the doorbell update
5	NMA starting to access the parameters and the queries
6	Cache coherency activity moves parameters and queries from the Host to the IKS
7	Offload starts and NMAs perform similarity search in parallel
8	NMA write partial top-K lists to the context output buffers
9	NMA updates the context doorbell
10	Doorbell cacheline moves from the IKS to the Host cache
11	CPU resumes and reads the context output buffers

Fig. 6. CPU-IKS interface through cache coherent CXL interconnect.

As IKS uses a scale-out near-memory processing architecture (§5.1), the embedding vectors are distributed across different near-memory accelerators' local DRAM. Therefore, after all the near-memory accelerators complete the offload, the CPU process waiting on `umwait()` will be notified and execute an aggregation routine to construct a single top-K list. This top-K list is then used to retrieve the actual top-K documents from the host memory. The CPU will locate documents based on the physical addresses of the top-K embedding vectors, as the addresses of the embedding vectors stored in IKS are known a priori.

5.3 Cache Coherent Interface

IKS leverages the cache-coherent interconnect in CXL.cache to implement an efficient interface between near-memory accelerators and host processes through shared memory. Figure 6 illustrates the transactions through the CXL.cache interface between the host and IKS to initiate and conclude an offload. The host process writes the *offload context* to the predefined *context buffer* address range shared between NMAs and the host CPU (step 1). Note that the context buffer is cacheable, and the CPU uses temporal writes to populate the buffers. Next, the host process writes into a doorbell register, which is mapped to a cache line shared by NMAs. NMAs poll on the doorbell register, and as soon as there is a change, the offload starts (step 4). Once the host updates the doorbell register, it calls `umwait()` to monitor the register for changes from the IKS side.

Before computation in the NMA can start, the NMA reads the offload context from the IKS cache (step 5) and the context written by the host is moved to NMA's scratchpad. Once the NMA computation is complete, the NMA updates the context buffers with the partial list of similarity scores and physical addresses of the corresponding embedding vectors. Lastly, the NMA writes into the doorbell register, and the host gets notified of the completion of the offload through the `umwait()` and `monitor()` mechanisms (step 11).

Our experimental results on a two-socket Sapphire Rapids CPU show that communicating the offload context through cache-coherent shared memory provides 1.6× higher throughput compared with using non-temporal writes that mimic the PCIe MMIO datapath (i.e., CXL.io). Using a cache-coherent interconnect to implement the notification mechanism through the producer/consumer-style doorbell register eliminates the need for expensive interrupt or polling mechanisms.

5.4 NMA Architecture

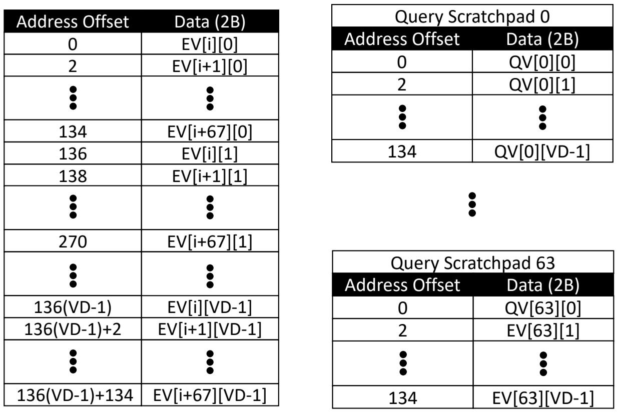
As shown in Figure 5c, each NMA implements 64 processing engines to accommodate similarity score calculations for up to 64 query vectors in parallel. Each processing engine includes a query scratchpad, dot-product unit, Top-K unit, and Top-K scratchpad. There is a central control unit in each NMA that generates memory accesses, controls data movement within the NMA, and activates processing engines based on the number of query vectors provided by the host CPU. The network-on-chip implements a fixed broadcast network from DRAM to all the processing engines to reuse data when multiple processing engines are active and evaluate similarity scores against different query vectors.

As shown in Figure 5d, the dot-product unit includes 68 MAC units, each operating at a 1 GHz frequency and providing 68 GFLOPS (16-bit floating point multiply-accumulate operations) compute throughput; therefore saturating the 136 GBps memory bandwidth of the LPDDR5X channels. Each MAC unit evaluates the similarity score between the query (stored in the query scratchpad) and an embedding vector that is read from DRAM in *VD* (Vector Dimension) cycles. All the processing engines operate on the same data that is read from the DRAM; in other words, each processing engine evaluates the similarity score between different query vectors and the same set of embedding vectors. Therefore, for a batch size of one, only one processing engine is utilized, and for a batch size of 64, all the processing engines are utilized. This way, we reuse the embedding vectors that are read from DRAM across different batch sizes.

As illustrated in Figure 5d, within an active dot-product unit, 68 MAC operations are performed in each clock cycle. The first input of the MAC units is element j of the query vector in processing engine PE (QV[PE][j]), and the second input is element j of the embedding vectors i to i+67 read from DRAM. As mentioned earlier, it takes *VD* (Vector Dimension) cycles for a dot-product unit to evaluate the similarity score for a block of 68 embedding vectors. Once the similarity score is evaluated, it is loaded into a *score register* (shown in Figure 5d) in the next clock cycle, and the MAC unit gets busy evaluating a new similarity score for the next 68 embedding vector block. The score registers (68 per processing engine) are then streamed out to the Top-K unit in the next 68 clock cycles.

The Top-K unit maintains an ordered list of the scores by comparing the incoming similarity scores with the head of the ordered list. Figure 5e illustrates the Top-K unit. If the value of the incoming score is larger, it is ignored; otherwise, it is inserted into the ordered list. Because the vector dimensions are much larger than 68, the serialized insertion into the ordered list is overlapped with the similarity score evaluations and is not on the critical path of the NMA offload.

After all the embedding vectors stored in the DRAM are evaluated, the control unit signals the end of the offload by



Address Offset	Data (2B)
0	EV[i][0]
2	EV[i+1][0]
⋮	⋮
134	EV[i+67][0]
136	EV[i][1]
138	EV[i+1][1]
⋮	⋮
270	EV[i+67][1]
⋮	⋮
136(VD-1)	EV[i][VD-1]
136(VD-1)+2	EV[i+1][VD-1]
⋮	⋮
136(VD-1)+134	EV[i+67][VD-1]

(a) DRAM

Query Scratchpad 0

Address Offset	Data (2B)
0	QV[0][0]
2	QV[0][1]
⋮	⋮
134	QV[0][VD-1]

⋮

Query Scratchpad 63

Address Offset	Data (2B)
0	QV[63][0]
2	EV[63][1]
⋮	⋮
134	EV[63][VD-1]

(b) Query Scratchpads

Fig. 7. Data layout and mappings: (a) DRAM mapping and (b) query scratchpads. EV, QV, and VD stand for Embedding Vector, Query Vector, and Vector Dimension, respectively.

loading the ordered Top-K list into the Top-K scratchpad and writing to the doorbell register. The host CPU is then notified and can read the content of the Top-K scratchpad through the CXL.cache protocol. Note that both the query scratchpad and the Top-K scratchpad are mapped to the host memory address space. In the current incarnation of the NMA, the size of the query scratchpad (per processing engine) is 2KB, and we keep an ordered list of 32 scores (i.e., we set K to 32 in the hardware).

5.5 Data Layout Inside DRAM and Query Scratchpad

Figure 7 illustrates the arrangement of embedding vectors within the DRAM, as well as the organization of query vectors in each query scratchpad of the processing engines for a batch size of 64 (i.e., when all the processing engines are active). The host CPU is required to store the embedding vectors in blocks of 68 vectors, laid out in the DRAM as shown in Figure 7a. Because each embedding vector element is 2 bytes (16-bit floating point), each block is stored in $136 \times VD$ bytes within DRAM, where VD is the vector dimension. Within a block, the embedding vectors are stored in column-major order. This layout allows for efficient batching of corpus vectors, as each may be read and processed dimension-by-dimension. Consequently, each NMA will access up to 136 bytes per cycle from the memory controller read queue, comprising one element from 68 distinct embedding vectors.

As discussed in Section 5.3, the host CPU will fill the query scratchpads with query vectors before an offload starts. The query vectors are stored in sequential addresses within the query scratchpads, as illustrated in Figure 7b.

This proposed data layout inside DRAM and query scratchpads simplifies the address generation as well as the network-on-chip architecture of the NMAs. We modified the memory allocation scheme in the vector database application to implement the block data mapping of embedding vectors inside IKS DRAM as shown in Figure 7a.

Platform	Parameter	Description
CPU	CPU model	Intel Xeon 4416+ 16 cores @ 2.00 GHz
	L1 Cache	48 kB dcache, 32kB icache
	L2 Cache	2MB
	L3 Cache	37.5 MB shared
	AVX	2x AVX-512 FMA units (164 GFlop/s/core)
	OS	Ubuntu 22.04.3
	Kernel	Linux 5.15.0-88-generic
	Memory	512 GB DDR5-4000 across 8 channels (256 GB/s)
AMX	–	Intel AMX (BFloat16, 500 GFlop/s/core)
IKS	–	1.1 TB/s, 69.9 TFlop/s
GPU	GPU Model	NVIDIA H100 SXM: 3.35 TB/s, 1979 TFlop/s

Table 2. Processing Element Options. Memory configuration for Intel AMX is the same as for CPU.

6 Experimental Results

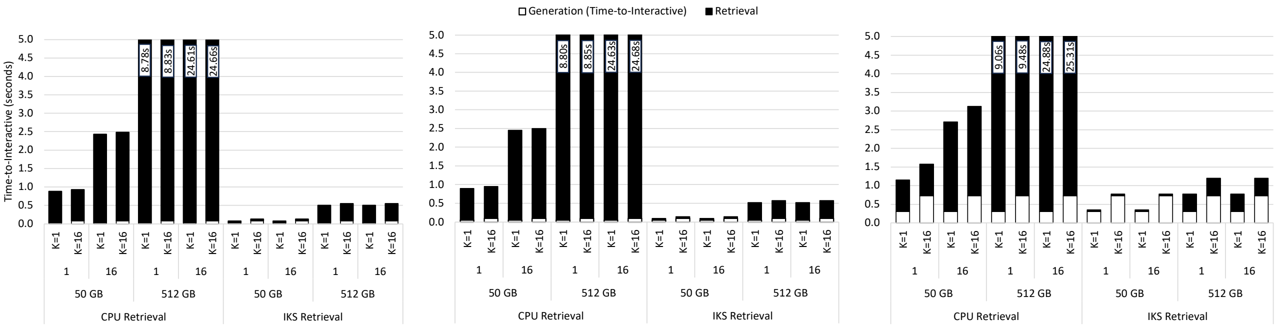
6.1 Methodology

To evaluate the performance of the IKS, we developed a simulator[3] and fed ENNS traces into it to obtain the retrieval time of IKS. The simulator is a cycle-approximate performance model that utilizes timing parameters from the RTL synthesis, LPDDR5X access timing, PCIe/CXL timing [47, 83], along with calculations of real software stack overhead (top-K aggregation and `umwait()` overhead). It emulates an IKS as a CXL device running on a remote CPU socket. We implemented the end-to-end RAG application described in Section 3 (i.e., `FiDT5`, `L3-8B`, and `L3-70B`), including the APIs for distributing queries to the NMA query scratchpad and reducing partial top-32 lists on the CPU. We ran the experiments on two servers equipped with Intel Xeon 4th generation CPUs and one NVIDIA H100 GPU NVIDIA GPUs. The system configuration is shown in Table 2.

We implemented the RTL design of the Near-Memory Accelerator (NMA) used in IKS and synthesized it using Synopsys Design Compiler targeting TSMC's 16nm technology node. This process involved collecting key metrics such as area, power, and timing to ensure the design meets the optimal criteria for operation at 1 GHz. For other components, we estimated the area of the memory controllers and PHYs based on die shots from the Apple M2 chip, which utilizes LPDDR5 in a 5nm process [2]. Since the area scaling of mixed-signal components is negligible [27, 90], we assumed the same area for the LPDDR5X PHYs and memory controllers when scaling to 16nm technology.

We developed a power model by evaluating the energy consumption of processing operations at the RTL level and incorporating the energy required for data access to scratchpads and LPDDR memory. For example, accessing data in SRAM consumes 39 fJ per bit, while LPDDR memory access requires 4 pJ per bit [16]. Since these energy values depend on the underlying technology node, we scaled them to correspond to a 16nm technology node for consistency [81].

[3] https://github.com/architecture-research-group/iks_simulator



(a) FiDT5 **(b)** L3-8B **(c)** L3-70B

Fig. 8. Inference time breakdown of CPU vs. IKS retrieval for FiDT5, L3-8B, and L3-70B. Generative model runs on GPU.

6.2 Software configuration

Google's Natural Questions (NQ) dataset [42, 43] is used for the evaluation of models. Meta's KILT benchmark [18] divides these into training (*nq-train*) and validation (*nq-dev*) datasets. For the retrieval phase, we use a BERT base (uncased) model trained to perform similarity searches between questions and their supporting documents in *nq-train*. The document corpus is constructed as described in [34], and an index is created using Faiss [31] to perform the similarity search[4]. Across ENNS and ANNS, Faiss is used for index management. The only change made in our evaluation is the use of Intel's OneMKL BLAS backend for ENNS for all batch sizes, as this provided better performance than the default Faiss search scheme, which uses only BLAS for batch sizes 20 and above.

FiDT5 Application: For testing the accuracy of FiDT5, as described in [29], the generator is initialized as a pretrained T5-base model (220 million parameters), then fine-tuned to predict answers from question-evidence pairs in the *nq-train* dataset.

To evaluate FiDT5 on the *nq-dev* dataset, we use the exact match metric [71], which normalizes answers and compares them against a list of acceptable answers. For FiDT5, *generation accuracy* scores refer to the percentage of *nq-dev* questions for which the RAG application generates a correct answer based on this exact match criterion.

L3-8B and L3-70B Applications: To evaluate L3-8B and L3-70B on the *nq-dev* dataset, we guide the model via prompting and evaluate *generation accuracy* using a Rouge-L "recall" metric [51], which scores answer predictions based on the proportion of the correct answer that is continuously present in the predicted answer. The model is instructed to give a short answer and to answer only if it is "completely sure." The prompting approach is used over fine-tuning to reflect an implementation that preserves the generality of the models. However, the downside of this approach is that evaluation is limited by prompt adherence, which is why the "recall" metric is used over precision or F1-Score.

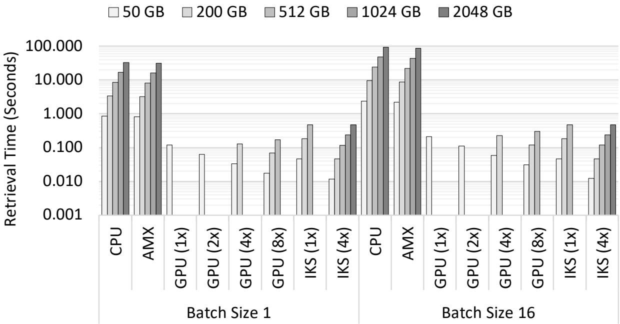


Fig. 9. Comparison of ENNS retrieval time for CPU, AMX, GPU (1, 2, 4, and 8 devices), and IKS (1, and 4 devices) for various corpus sizes. The absence of bars in specific GPU and IKS configurations indicates that the corpus exceeds the capacity of the accelerator memory. The Y-axis is in log-scale.

6.3 Effectiveness and Scalability of IKS Retrieval

Figure 9 compares the performance of IKS with CPU, AMX (idealized, based on speedup for matrix multiplication), and GPU ENNS retrieval. IKS provisions compute and memory bandwidth to balance the pipeline at the maximum batch size of 64; as such, performance is almost flat for batch sizes less than 64. As shown, the purposefully built NMA logic for ENNS enables 1 IKS unit to outperform 1 GPU for a 50GB corpus for batch sizes 1 and 16 by 2.6× and 4.6×, respectively. This counterintuitive speedup of IKS over GPUs, which theoretically have both higher FLOPS and memory bandwidth than IKS, is due to two reasons: (1) top-K tracking and aggregation on GPUs is not efficient, while IKS includes specialized Top-K units; and (2) low utilization of the GPU chip translates to limited memory bandwidth usage, as saturating the entire HBM memory bandwidth requires many streaming multiprocessors and tensor cores to issue memory accesses to DRAM in parallel.

To demonstrate the scalability of IKS, we include the retrieval time of multi-GPU and multi-IKS setups. Because each H100 GPU can fit 80GB of embedding vectors, 8 GPUs can accommodate maximum corpus size of 640GB. However, with only four IKS devices, we can fit up to a 2TB corpus size. As shown in Figure 9, with additional GPUs and IKS units, the

[4] We adapt the Faiss implementation of ENNS by using Intel MKL as the BLAS backend and increasing the corpus block size from 1024 to 16384.

Corpus Size	50 GB		512 GB	
Batch Size	1	64	1	64
Write Query Vector	0.3 us	1 us	0.3 us	1 us
Dot-Product	45.96 ms	45.96 ms	470.6ms	470.6 ms
Partial Top-32 Read	0.7 us	22.4 us	0.7 us	22.4 us
Top-K Aggregation	19 us	540 us	23 us	390 us
Total	46.0 ms	46.5 ms	470.6 ms	471.0 ms

Table 3. Breakdown of ENNS latency on IKS.

retrieval time for the same corpus size decreases, demonstrating the high data-level parallelism of ENNS and the strong scaling of both GPU and IKS. For example, GPU retrieval time for a 50GB corpus size reduces by 1.9×, 3.6×, and 6.9× with 2, 4, and 8 GPU devices, respectively, and IKS retrieval time for a 50GB corpus size reduces by 1× and 3.9× with 1 and 4 IKS units, respectively. Due to the low-overhead IKS-CPU interface, the dominance of similarity search latency in end-to-end ENNS retrieval, and the highly parallelizable nature of ENNS, IKS also provides near-perfect weak scaling. For instance, the retrieval time for a 2TB corpus on 4 IKS units is only 100μs longer than for a 512GB corpus on 1 IKS unit. However, we do not evaluate configurations with more than four IKS units, and the overhead of host-side final top-K aggregation scales as additional units are added. Additionally, we do not evaluate deployments of IKS spanning multiple nodes.

Table 3 reports the absolute time breakdown of ENNS retrieval on IKS. We break down the retrieval time of IKS into four components: transfer time of query vectors over the CXL interconnect to the NMAs, time for performing dot-products (both computation and DRAM accesses), updating the top-k score lists in parallel on all NMAs, and time for reducing the partial top-32 lists into a single one on the CPU. The retrieval time of IKS does not change with the value of K (with a maximum K value of 32). This is because IKS always returns 32 top similarity scores, and it is up to the retriever model to pass between 1 to 32 of them to the generative model. As shown in the table, the majority of time is spent on computations and DRAM accesses, and the overhead of initiating offload over the cache-coherent interconnect and aggregating top-K documents on the CPU is negligible.

6.4 End-to-End Performance

Figure 8 compares the end-to-end inference time of `FiDT5`, `L3-8B`, and `L3-70B` when CPU and IKS are used for ENNS retrieval for various batch sizes, document counts, and corpus sizes. As shown, for large corpus sizes or large batch sizes, the inference time of the RAG applications with CPU retrieval exceeds several seconds, which is not acceptable for user-facing question-answering applications. IKS significantly reduces the ENNS retrieval time for the applications. The end-to-end inference time speedup provided by IKS ranges between 5.7 to 67× for `FiDT5`, between 1.8 and 15.5× for `L3-8B`, and between 1.1 and 9.7× for `L3-70B` for various batch sizes, corpus sizes, and document counts.

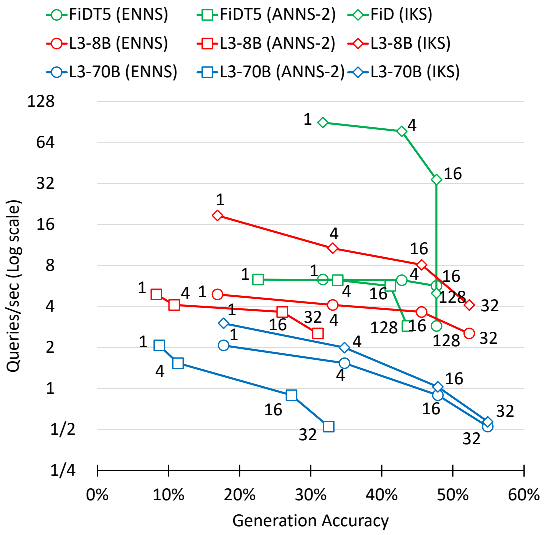


Fig. 10. Comparison of accuracy and throughput of `FiDT5`,`L3-8B`,and `L3-70B` for various configurations. The data labels represent passage count. ANNS-2 is an HNSW index with *M*, *efConstruction*, and *efSearch* of 64, 128, and 10000, respectively.

To gain a comprehensive understanding of how the performance and accuracy of RAG applications with IKS acceleration compare across various configurations, Figure 10 depicts the queries per second and accuracy of `FiDT5`, `L3-8B`, and `L3-70B` implemented using four different configurations: RAG with ENNS running on CPU, RAG with ANNS (two configurations) running on CPU, and RAG with ENNS running on IKS. The generative model runs on the GPU in all these configurations. As illustrated in Figure 10, although ANNS-2 configurations exhibit higher throughput compared to ENNS (running on the CPU), their accuracy is lower. For RAG applications that use IKS, retrieval is not a bottleneck, and throughput is significantly improved, even compared to ANNS, as the same generation accuracy can be achieved with smaller values of K (i.e., smaller but more accurate context sent to the generative model).

6.5 Power and Area Analysis

The area of each NMA, which contains 64 processing engines, each comprising a dot-product unit, a 2K SRAM query scratchpad, a top-K unit, and a top-K scratchpad, is approximately 3.4 mm^2 in the 16nm technology node. Additionally, 14 mm^2 is required for the PHYs and memory controllers. However, the area of the NMA chip is determined by the shoreline because the 21 mm of shoreline required per NMA (20 mm for the LPDDR5X PHYs and 1 mm for PCIe PHYs §5.1) necessitates that the NMA occupy at least 27.56 mm^2 in the 16nm technology node. The NMA can be manufactured using older technology nodes to reduce costs and prevent area wastage, as the PHY area (which is mixed-signal) does not scale at the same rate as SRAM and logic [27, 90].

For a batch size of 1 and vector dimensions of 1024, the processing engines, along with the corresponding query scratchpad accesses, consume approximately 59 mW, while accessing embedding vectors from LPDDR memory requires 4.35 W. As a result, the total power consumption of IKS for a batch size of 1 is 35.2 W. With larger batch sizes, data reuse ensures that the power required for LPDDR access remains constant, but the power consumption of the processing engines increases linearly as more engines are activated to handle the additional workload. For instance, at full utilization with a batch size of 64, the total power consumption increases to 65 W.

6.6 Cost and Power Comparison with GPU

IKS utilizes LPDDR5X memory to store embedding vectors. While figures for the cost of LPDDR5X are not yet available, we assume that HBM is more than 3× more expensive than LPDDR [58]. Since a single IKS unit includes 6.4× as much onboard memory as a single NVIDIA H100 GPU, the memory cost of IKS is expected to be approximately 2.5× greater than that of a GPU.

For the comparison of compute unit cost, the GPU has a die area of 826 mm^2, while the IKS NMAs total a die area of 220 mm^2. Because the production cost of a chip increases superlinearly with die area [59], an IKS unit (with 5× larger memory capacity) is expected to cost a fraction of a GPU.

7 Discussion

IKS provides a cost-effective solution for accelerating ENNS, where the quality of the search is not dataset-dependent. However, if the dataset is amenable to clustering, then the accuracy gap between ENNS and ANNS would reduce, making ANNS more attractive for retrieval. Moreover, IKS is best-suited to RAG applications requiring very high recall, and for datasets difficult to search with existing ANNS schemes with relatively large batch sizes. For example, modern ANNS schemes cannot eliminate more than 99% of the search space for the GloVe dataset [96], so at least 64% of the corpus must be read by an ANNS that doesn't offer data re-use across queries; in which case the overheads of common ANNS schemes reduces performance to below that of ENNS. However, for datasets that are easier to filter, there is an opportunity for improvement by incorporating approximation techniques into IKS; however, this introduces significant challenges as IKS owes much of its performance to the sequential memory access pattern of ENNS.

One key inefficiency of IKS is that it performs an exhaustive search over the entire corpus, which consumes energy and saturates memory bandwidth. The high internal memory bandwidth utilization of ENNS can cause slowdowns for external accesses by other applications that use IKS as a memory expander, rather than a vector database accelerator. Exploring early termination of similarity search [12, 46] could be a natural solution for reducing the memory bandwidth utilization of ENNS without compromising search accuracy.

Another inefficiency in the current version of IKS is the low NMA chip utilization for batch sizes less than 64. The rationale for overprovisioning NMA compute is that we effectively have free area on the NMA chip. Note that each NMA chip requires eight LPDDR5X memory channels, which demand 20mm of chip shoreline. Therefore, the minimum NMA chip area is 25mm^2 (§Section 5.1). Thus, the area on NMA is effectively free up to a cap of 25mm^2. We chose to utilize this "free" area to overprovision compute so that IKS remains memory-bandwidth bound for all batch sizes below 64. There are opportunities for circuit-level techniques, such as clock and power gating, to power off extra processing engines when the batch size is below 64. Moreover, dynamic voltage and frequency scaling can be used to reduce the frequency and voltage of the NMA chip for batch sizes less than 64, allowing multiple processing engines to perform similarity searches for each query vector.

8 Related Work

Sim et al. [86] implement a computational CXL memory solution for near-memory processing and showcased ENNS acceleration inside the CXL memory. However, this work implements CXL memory using DDR DRAM, which does not meet the power and bandwidth requirements for ENNS on large corpus sizes used in RAG. Additionally, our work implements a novel interface between host and near-memory accelerators through CXL.cache. Lee et al. [44] and Wang et al. [97] present near-data accelerators for PQ- and Graph-based ANNS, respectively. However, we accelerate ENNS because different corpora are amenable to different ANNS algorithms, and the complex algorithms and memory access patterns of such ANNS schemes also make ANNS accelerators highly task-specific. Ke et al. [37] propose near-memory acceleration of DLRM on Samsung AxDIMM. AxDIMM is based on a DIMM form factor that limits per-rank memory capacity and compromises the memory capacity of the host CPU when used as an accelerator (§4). In contrast, IKS does not strand the internal DRAM space and does not have capacity or compute throughput limitations.

Concurrent with our work, others have also observed that low-quality retrieval can lead to both low-quality and slow generation. Corrective RAG filters out irrelevant documents from the retrieved list before sending them to the LLM [104], while Sparse RAG enables LLMs to use only highly relevant retrieved information [112]. In this work, we used ENNS to eliminate the risk of low-quality retrieval and reduce the context size.

9 Conclusion

In this work, we profiled representative RAG applications and showed that the retrieval phase can be an accuracy, latency, and throughput bottleneck, highlighting the importance of an exact, yet high-performance and scalable retrieval scheme for future RAG applications. We designed, implemented, and evaluated the Intelligent Knowledge Store (IKS), a CXL-type-2 device for near-memory acceleration of exact K nearest neighbor search.

The key novelty of IKS is the hardware/software co-design that enables a scale-out near-memory processing architecture by leveraging cache-coherent shared memory between the CPU and near-memory accelerators. IKS offers 18-52× faster exact nearest neighbor search over a 512 GB vector database compared to executing the search on Intel Sapphire Rapids accelerators, leading to 2.0-49× lower end-to-end RAG inference time.

Acknowledgments

This work was supported in part by NSF grant numbers 2239020, 1565570, and 2402873, in part by ACE, one of the seven centers in JUMP 2.0, a Semiconductor Research Corporation (SRC) program sponsored by DARPA, in part by the Office of Naval Research contract number N000142412612, and in part by the Center for Intelligent Information Retrieval. Any opinions, findings, conclusions, and recommendations expressed in this material are those of the authors and do not necessarily reflect those of the sponsors. We thank Jae-sun Seo and Yuan Liao from Cornell University for their help in synthesizing the near-memory accelerators on 16nm TSMC technology.

References

[1] The Shift from Models to Compound AI Systems. *Berkeley Artificial Intelligence Research Blog*. URL https://bair.berkeley.edu/blog/2024/02/18/compound-ai-systems/.

[2] Die analysis: Samsung exynos 2200 with rdna2 graphics. *Locuza (substack)*, 2022. URL https://locuza.substack.com/p/die-analysis-samsung-exynos-2200.

[3] Mask / Reticle. *Wikichip*, 2024. URL https://en.wikichip.org/wiki/mask#:~:text=Reticle%20limit%5Bedit%5D,use%20of%20anamorphic%20lens%20array.

[4] Cxl is dead in the ai era. *Semianalysis*, 2024. URL https://www.semianalysis.com/p/cxl-is-dead-in-the-ai-era.

[5] Amey Agrawal, Ashish Panwar, Jayashree Mohan, Nipun Kwatra, Bhargav S. Gulavani, and Ramachandran Ramjee. Sarathi: Efficient llm inference by piggybacking decodes with chunked prefills. *Microsoft Research Blog*, September 2023. URL https://doi.org/10.48550/arXiv.2308.16369.

[6] Yeonchan Ahn, Sang-Goo Lee, Junho Shim, and Jaehui Park. Retrieval-augmented response generation for knowledge-grounded conversation in the wild. *IEEE Access*, 10:131374–131385, 2022. doi:10.1109/ACCESS.2022.3228964. URL https://doi.org/10.1109/ACCESS.2022.3228964.

[7] Mohammad Alian, Seung Won Min, Hadi Asgharimoghaddam, Ashutosh Dhar, Dong Kai Wang, Thomas Roewer, Adam McPadden, Oliver O'Halloran, Deming Chen, Jinjun Xiong, et al. Application-transparent near-memory processing architecture with memory channel network. In *2018 51st Annual IEEE/ACM International Symposium on Microarchitecture (MICRO)*, pages 802–814. IEEE, 2018.

[8] Keivan Alizadeh, Iman Mirzadeh, Dmitry Belenko, Karen Khatamifard, Minsik Cho, Carlo C Del Mundo, Mohammad Rastegari, and Mehrdad Farajtabar. Llm in a flash: Efficient large language model inference with limited memory. *arXiv preprint arXiv:2312.11514*, 2023. doi:10.48550/arXiv.2312.11514. URL https://doi.org/10.48550/arXiv.2312.11514.

[9] Martin Aumüller, Erik Bernhardsson, and Alexander Faithfull. Ann-benchmarks: A benchmarking tool for approximate nearest neighbor algorithms. *Information Systems*, 87:101374, 2020. URL https://doi.org/10.48550/arXiv.1807.05614.

[10] Artem Babenko and Victor Lempitsky. The inverted multi-index. In *2012 IEEE Conference on Computer Vision and Pattern Recognition*, pages 3069–3076, 2012. doi:10.1109/CVPR.2012.6248038. URL https://doi.org/10.1109/CVPR.2012.6248038.

[11] Giovanni Bonetta, Rossella Cancelliere, Ding Liu, and Paul Vozila. Retrieval-augmented transformer-xl for close-domain dialog generation. *The International FLAIRS Conference Proceedings*, 34(1), April 2021. ISSN 2334-0762. doi:10.32473/flairs.v34i1.128369. URL http://dx.doi.org/10.32473/flairs.v34i1.128369.

[12] Francesco Busolin, Claudio Lucchese, Franco Maria Nardini, Salvatore Orlando, Raffaele Perego, and Salvatore Trani. Early exit strategies for approximate k-nn search in dense retrieval. In *Proceedings of the 33rd ACM International Conference on Information and Knowledge Management*, CIKM '24, page 3647–3652, New York, NY, USA, 2024. Association for Computing Machinery. ISBN 9798400704369. doi:10.1145/3627673.3679903. URL https://doi.org/10.1145/3627673.3679903.

[13] Deng Cai, Yan Wang, Wei Bi, Zhaopeng Tu, Xiaojiang Liu, Wai Lam, and Shuming Shi. Skeleton-to-response: Dialogue generation guided by retrieval memory. In Jill Burstein, Christy Doran, and Thamar Solorio, editors, *Proceedings of the 2019 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies, Volume 1 (Long and Short Papers)*, pages 1219–1228, Minneapolis, Minnesota, June 2019. Association for Computational Linguistics. doi:10.18653/v1/N19-1124. URL https://doi.org/10.18653/v1/N19-1124.

[14] Wenhu Chen, Hexiang Hu, Xi Chen, Pat Verga, and William Cohen. MuRAG: Multimodal retrieval-augmented generator for open question answering over images and text. In Yoav Goldberg, Zornitsa Kozareva, and Yue Zhang, editors, *Proceedings of the 2022 Conference on Empirical Methods in Natural Language Processing*, pages 5558–5570, Abu Dhabi, United Arab Emirates, December 2022. Association for Computational Linguistics. doi:10.18653/v1/2022.emnlp-main.375. URL https://aclanthology.org/2022.emnlp-main.375.

[15] Wenhu Chen, Hexiang Hu, Chitwan Saharia, and William W. Cohen. Re-imagen: Retrieval-augmented text-to-image generator. *ArXiv*, abs/2209.14491, 2022. URL https://api.semanticscholar.org/CorpusID:252596087.

[16] William J. Dally, Yatish Turakhia, and Song Han. Domain-specific hardware accelerators. *Communications of the ACM*, 63(7):48–57, 2020. doi:10.1145/3361682. URL https://doi.org/10.1145/3361682.

[17] Feyza Duman Keles, Pruthuvi Mahesakya Wijewardena, and Chinmay Hegde. On the computational complexity of self-attention. In Shipra Agrawal and Francesco Orabona, editors, *Proceedings of The 34th International Conference on Algorithmic Learning Theory*, volume 201 of *Proceedings of Machine Learning Research*, pages 597–619. PMLR, 20 Feb–23 Feb 2023. URL https://proceedings.mlr.press/v201/duman-keles23a.html.

[18] Angela Fan Fabio Petroni, Aleksandra Piktus. Introducing KILT, a new unified benchmark for knowledge-intensive NLP tasks — ai.meta.com. *Meta AI Blog*. URL https://ai.meta.com/blog/introducing-kilt-a-new-unified-benchmark-for-knowledge-intensive-nlp-tasks/. [Accessed 22-11-2023].

[19] Zhengcong Fei. Memory-augmented image captioning. *Proceedings of the AAAI Conference on Artificial Intelligence*, 35 (2):1317–1324, May 2021. doi:10.1609/aaai.v35i2.16220. URL https://ojs.aaai.org/index.php/AAAI/article/view/16220.

[20] Amin Firoozshahian, Joel Coburn, Roman Levenstein, Rakesh Nattoji, Ashwin Kamath, Olivia Wu, Gurdeepak Grewal, Harish Aepala, Bhasker Jakka, Bob Dreyer, Adam Hutchin, Utku Diril, Krishnakumar Nair, Ehsan K. Aredestani, Martin Schatz, Yuchen Hao, Rakesh Komuravelli, Kunming Ho, Sameer Abu Asal, Joe Shajrawi, Kevin Quinn, Nagesh Sreedhara, Pankaj Kansal, Willie Wei, Dheepak Jayaraman, Linda Cheng, Pritam Chopda, Eric Wang, Ajay Bikumandla, Arun

Karthik Sengottuvel, Krishna Thottempudi, Ashwin Narasimha, Brian Dodds, Cao Gao, Jiyuan Zhang, Mohammed Al-Sanabani, Ana Zehtabioskuie, Jordan Fix, Hangchen Yu, Richard Li, Kaustubh Gondkar, Jack Montgomery, Mike Tsai, Saritha Dwarakapuram, Sanjay Desai, Nili Avidan, Poorvaja Ramani, Karthik Narayanan, Ajit Mathews, Sethu Gopal, Maxim Naumov, Vijay Rao, Krishna Noru, Harikrishna Reddy, Prahlad Venkatapuram, and Alexis Bjorlin. Mtia: First generation silicon targeting meta's recommendation systems. In *Proceedings of the 50th Annual International Symposium on Computer Architecture*, ISCA '23, New York, NY, USA, 2023. Association for Computing Machinery. ISBN 9798400700958. doi:10.1145/3579371.3589348. URL https://doi.org/10.1145/3579371.3589348.

[21] In Gim, Guojun Chen, Seung-seob Lee, Nikhil Sarda, Anurag Khandelwal, and Lin Zhong. Prompt cache: Modular attention reuse for low-latency inference. *arXiv preprint arXiv:2311.04934*, 2023. URL https://doi.org/10.48550/arXiv.2311.04934.

[22] Jiatao Gu, Yong Wang, Kyunghyun Cho, and Victor OK Li. Search engine guided neural machine translation. In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 32, 2018. URL https://doi.org/10.1609/aaai.v32i1.12013.

[23] Liangke Gui, Borui Wang, Qiuyuan Huang, Alexander Hauptmann, Yonatan Bisk, and Jianfeng Gao. KAT: A knowledge augmented transformer for vision-and-language. In Marine Carpuat, Marie-Catherine de Marneffe, and Ivan Vladimir Meza Ruiz, editors, *Proceedings of the 2022 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies*, pages 956–968, Seattle, United States, July 2022. Association for Computational Linguistics. doi:10.18653/v1/2022.naacl-main.70. URL https://aclanthology.org/2022.naacl-main.70.

[24] Tatsunori B. Hashimoto, Kelvin Guu, Yonatan Oren, and Percy Liang. A retrieve-and-edit framework for predicting structured outputs. In *Proceedings of the 32nd International Conference on Neural Information Processing Systems*, NIPS'18, page 10073–10083, Red Hook, NY, USA, 2018. Curran Associates Inc. URL https://dl.acm.org/doi/10.5555/3327546.3327670.

[25] Qiuxiang He, Guoping Huang, Qu Cui, Li Li, and Lemao Liu. Fast and accurate neural machine translation with translation memory. In Chengqing Zong, Fei Xia, Wenjie Li, and Roberto Navigli, editors, *Proceedings of the 59th Annual Meeting of the Association for Computational Linguistics and the 11th International Joint Conference on Natural Language Processing (Volume 1: Long Papers)*, pages 3170–3180, Online, August 2021. Association for Computational Linguistics. doi:10.18653/v1/2021.acl-long.246. URL https://aclanthology.org/2021.acl-long.246.

[26] Sebastian Hofstätter, Jiecao Chen, Karthik Raman, and Hamed Zamani. Fid-light: Efficient and effective retrieval-augmented text generation. In *Proceedings of the 46th International ACM SIGIR Conference on Research and Development in Information Retrieval*, SIGIR '23, page 1437–1447, New York, NY, USA, 2023. Association for Computing Machinery. ISBN 9781450394086. doi:10.1145/3539618.3591687. URL https://doi.org/10.1145/3539618.3591687.

[27] Mark Horowitz. 1.1 computing's energy problem (and what we can do about it). In *2014 IEEE International Solid-State Circuits Conference Digest of Technical Papers (ISSCC)*, pages 10–14, 2014. doi:10.1109/ISSCC.2014.6757323. URL https://doi.org/10.1109/ISSCC.2014.6757323.

[28] Mohamed Assem Ibrahim, Onur Kayiran, Yasuko Eckert, Gabriel H Loh, and Adwait Jog. Analyzing and leveraging decoupled l1 caches in gpus. In *2021 IEEE International Symposium on High-Performance Computer Architecture (HPCA)*, pages 467–478. IEEE, 2021.

[29] 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. doi:10.18653/v1/2021.eacl-main.74. URL https://aclanthology.org/2021.eacl-main.74.

[30] Gautier Izacard and Edouard Grave. Distilling knowledge from reader to retriever for question answering. In *International Conference on Learning Representations*, 2021. URL https://openreview.net/forum?id=NTEz-6wysdb.

[31] Hervé Jegou, Matthijs Douze, and Jeff Johnson. Faiss: A library for efficient similarity search — engineering.fb.com. *Meta Engineering Blog*. URL https://engineering.fb.com/2017/03/29/data-infrastructure/faiss-a-library-for-efficient-similarity-search/. [Accessed 12-11-2023].

[32] Herve Jégou, Matthijs Douze, and Cordelia Schmid. Product quantization for nearest neighbor search. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 33 (1):117–128, 2011. doi:10.1109/TPAMI.2010.57. URL https://doi.org/10.1109/TPAMI.2010.57.

[33] Nikhil Kandpal, Eric Wallace, and Colin Raffel. Deduplicating training data mitigates privacy risks in language models. In *International Conference on Machine Learning*, pages 10697–10707. PMLR, 2022. URL https://proceedings.mlr.press/v162/kandpal22a.html.

[34] Vladimir Karpukhin, Barlas Oguz, Sewon Min, Patrick Lewis, Ledell Wu, Sergey Edunov, Danqi Chen, and Wen-tau Yih. Dense passage retrieval for open-domain question answering. In Bonnie Webber, Trevor Cohn, Yulan He, and Yang Liu, editors, *Proceedings of the 2020 Conference on Empirical Methods in Natural Language Processing (EMNLP)*, pages 6769–6781, Online, November 2020. Association for Computational Linguistics. doi:10.18653/v1/2020.emnlp-main.550. URL https://aclanthology.org/2020.emnlp-main.550.

[35] Jungo Kasai, Keisuke Sakaguchi, Yoichi Takahashi, Ronan Le Bras, Akari Asai, Xinyan Velocity Yu, Dragomir Radev, Noah A. Smith, Yejin Choi, and Kentaro Inui. Realtime qa: what's the answer right now? In *Proceedings of the 37th International Conference on Neural Information Processing Systems*, NIPS '23, Red Hook, NY, USA, 2024. Curran Associates Inc. URL https://dl.acm.org/doi/10.5555/3666122.3668252.

[36] Amirhossein Kazemnejad, Mohammadreza Salehi, and Mahdieh Soleymani Baghshah. Paraphrase generation by learning how to edit from samples. In Dan Jurafsky, Joyce Chai, Natalie Schluter, and Joel Tetreault, editors, *Proceedings of the 58th Annual Meeting of the Association for Computational Linguistics*, pages 6010–6021, Online, July 2020. Association for Computational Linguistics. doi:10.18653/v1/2020.acl-main.535. URL https://aclanthology.org/2020.acl-main.535.

[37] Liu Ke, Xuan Zhang, Jinin So, Jong-Geon Lee, Shin-Haeng Kang, Sukhan Lee, Songyi Han, YeonGon Cho, Jin Hyun Kim, Yongsuk Kwon, KyungSoo Kim, Jin Jung, Ilkwon Yun, Sung Joo Park, Hyunsun Park, Joonho Song, Jeonghyeon Cho, Kyomin Sohn, Nam Sung Kim, and Hsien-Hsin S. Lee. Near-memory processing in action: Accelerating personalized recommendation with axdimm. *IEEE Micro*, 42 (1):116–127, 2022. URL https://doi.org/10.1109/MM.2021.3097700.

[38] Ben Keller, Rangharajan Venkatesan, Steve Dai, Stephen G Tell, Brian Zimmer, Charbel Sakr, William J Dally, C Thomas Gray, and Brucek Khailany. A 95.6-tops/w deep learning inference accelerator with per-vector scaled 4-bit quantization in 5 nm. *IEEE Journal of Solid-State Circuits*, 58(4):1129–1141, 2023. URL https://doi.org/https://doi.org/10.1109/VLSITechnologyandCir46769.2022.9830277.

[39] Jin Hyun Kim, Shin-Haeng Kang, Sukhan Lee, Hyeonsu Kim, Yuhwan Ro, Seungwon Lee, David Wang, Jihyun Choi, Jinin So, YeonGon Cho, JoonHo Song, Jeonghyeon Cho, Kyomin Sohn, and Nam Sung Kim. Aquabolt-xl hbm2-pim, lpddr5-pim with in-memory processing, and axdimm with acceleration buffer. *IEEE Micro*, 42(3):20–30, 2022. doi:10.1109/MM.2022.3164651. URL https://doi.org/10.1109/MM.2022.3164651.

[40] To Eun Kim, Alireza Salemi, Andrew Drozdov, Fernando Diaz, and Hamed Zamani. Retrieval-enhanced machine learning: Synthesis and opportunities, 2024. URL https://arxiv.org/abs/2407.12982.

[41] Ishita Kumar, Snigdha Viswanathan, Sushrita Yerra, Alireza Salemi, Ryan A. Rossi, Franck Dernoncourt, Hanieh Deilamsalehy, Xiang Chen, Ruiyi Zhang, Shubham Agarwal, Nedim Lipka, Chien Van Nguyen, Thien Huu Nguyen, and Hamed Zamani. Longlamp: A benchmark for personalized long-form text generation, 2024. URL https://arxiv.org/abs/2407.11016.

[42] Tom Kwiatkowski, Jennimaria Palomaki, Olivia Redfield, Michael Collins, Ankur Parikh, Chris Alberti, Danielle Epstein, Illia Polosukhin, Matthew Kelcey, Jacob Devlin, Kenton Lee, Kristina N. Toutanova, Llion Jones, Ming-Wei Chang, Andrew Dai, Jakob Uszkoreit, Quoc Le, and Slav Petrov. Natural questions: a benchmark for question answering research. *Transactions of the Association of Computational Linguistics*, 2019. URL https://doi.org/10.1162/tacl_a_00276.

[43] Kenton Lee, Ming-Wei Chang, and Kristina Toutanova. Latent retrieval for weakly supervised open domain question answering. *arXiv preprint arXiv:1906.00300*, 2019. URL https://doi.org/10.48550/arXiv.1906.00300.

[44] Yejin Lee, Hyunji Choi, Sunhong Min, Hyunseung Lee, Sangwon Beak, Dawoon Jeong, Jae W. Lee, and Tae Jun Ham. Anna: Specialized architecture for approximate nearest neighbor search. In *2022 IEEE International Symposium on High-Performance Computer Architecture (HPCA)*, pages 169–183, 2022. doi:10.1109/HPCA53966.2022.00021.

[45] Patrick Lewis, Ethan Perez, Aleksandra Piktus, Fabio Petroni, Vladimir Karpukhin, Naman Goyal, Heinrich Küttler, Mike Lewis, Wen-tau Yih, Tim Rocktäschel, Sebastian Riedel, and Douwe Kiela. Retrieval-augmented generation for knowledge-intensive nlp tasks. In *Proceedings of the 34th International Conference on Neural Information Processing Systems*, NIPS'20, Red Hook, NY, USA, 2020. Curran Associates Inc. ISBN 9781713829546. URL https://doi.org/10.5555/3495724.3496517.

[46] Conglong Li, Minjia Zhang, David G. Andersen, and Yuxiong He. Improving approximate nearest neighbor search through learned adaptive early termination. In *Proceedings of the 2020 ACM SIGMOD International Conference on Management of Data*, SIGMOD '20, page 2539–2554, New York, NY, USA, 2020. Association for Computing Machinery. ISBN 9781450367356. doi:10.1145/3318464.3380600. URL https://doi.org/10.1145/3318464.3380600.

[47] 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. ISBN 9781450399166. URL https://doi.org/10.1145/3575693.3578835.

[48] Huayang Li, Yixuan Su, Deng Cai, Yan Wang, and Lemao Liu. A survey on retrieval-augmented text generation. *ArXiv*, abs/2202.01110, 2022. URL https://api.semanticscholar.org/CorpusID:246472929.

[49] Wen Li, Ying Zhang, Yifang Sun, Wei Wang, Mingjie Li, Wenjie Zhang, and Xuemin Lin. Approximate nearest neighbor search on high dimensional data — experiments, analyses, and improvement. *IEEE Transactions on Knowledge and Data Engineering*, 32 (8):1475–1488, 2020. doi:10.1109/TKDE.2019.2909204. URL https://doi.org/10.1109/TKDE.2019.2909204.

[50] Pierre Lienhart. Llm inference series: 4. kv caching, a deeper look. *Pierre Leinhart (Medium)*, Jan 2024. URL https://medium.com/@plienhar/llm-inference-series-4-kv-caching-a-deeper-look-4ba9a77746c8.

[51] Chin-Yew Lin. ROUGE: A package for automatic evaluation of summaries. In *Text Summarization Branches Out*, pages 74–81, Barcelona, Spain, July 2004. Association for Computational Linguistics. URL https://aclanthology.org/W04-1013.

[52] Ji Lin, Jiaming Tang, Haotian Tang, Shang Yang, Xingyu Dang, and Song Han. Awq: Activation-aware weight quantization for llm compression and acceleration. *arXiv preprint arXiv:2306.00978*, 2023. URL https://doi.org/10.48550/arXiv.2306.00978.

[53] Shangqing Liu, Yu Chen, Xiaofei Xie, Jing Kai Siow, and Yang Liu. Retrieval-augmented generation for code summarization via hybrid GNN. In *International Conference on Learning Representations*, 2021. URL https://openreview.net/forum?id=zv-typ1gPxA.

[54] Gabriel H. Loh, Natalie Enright Jerger, Ajaykumar Kannan, and Yasuko Eckert. Interconnect-memory challenges for multi-chip, silicon interposer systems. In *Proceedings of the 2015 International Symposium on Memory Systems*, MEMSYS '15, page 3–10, New York, NY, USA, 2015. Association for Computing Machinery. ISBN 9781450336048. doi:10.1145/2818950.2818951. URL https://doi.org/10.1145/2818950.2818951.

[55] Yu A. Malkov and D. A. Yashunin. Efficient and robust approximate nearest neighbor search using hierarchical navigable small world graphs. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 42(4):824–836, apr 2020. ISSN 0162-8828. doi:10.1109/TPAMI.2018.2889473. URL https://doi.org/10.1109/TPAMI.2018.2889473.

[56] Meta. Meta llama 3. *Meta*, 2024. URL https://llama.meta.com/llama3/. Accessed: 2024-06-18.

[57] Xupeng Miao, Gabriele Oliaro, Zhihao Zhang, Xinhao Cheng, Zeyu Wang, Zhengxin Zhang, Rae Ying Yee Wong, Alan Zhu, Lijie Yang, Xiaoxiang Shi, Chunan Shi, Zhuoming Chen, Daiyaan Arfeen, Reyna Abhyankar, and Zhihao Jia. Specinfer: Accelerating large language model serving with tree-based speculative inference and verification. In *Proceedings of the 29th ACM International Conference on Architectural Support for Programming Languages and Operating Systems, Volume 3*, ASPLOS '24, page 932–949, New York, NY, USA, 2024. Association for Computing Machinery. ISBN 9798400703867. doi:10.1145/3620666.3651335. URL https://doi.org/10.1145/3620666.3651335.

[58] Timothy Prickett Morgan. He who can pay top dollar for hbm memory controls ai training. *The Next Platform*, 2024. URL https://www.nextplatform.com/2024/02/27/he-who-can-pay-top-dollar-for-hbm-memory-controls-ai-training/. Accessed: 2024-06-23.

[59] Samuel Naffziger, Kevin Lepak, Milam Paraschou, and Mahesh Subramony. 2.2 amd chiplet architecture for high-performance server and desktop products. In *2020 IEEE International Solid- State Circuits Conference - (ISSCC)*, pages 44–45, 2020. doi:10.1109/ISSCC19947.2020.9063103.

[60] OpenAI. Chatgpt plugins. *OpenAI Blog*, 2023. URL https://openai.com/blog/chatgpt-plugins.

[61] Sang-Soo Park, KyungSoo Kim, Jinin So, Jin Jung, Jonggeon Lee, Kyoungwan Woo, Nayeon Kim, Younghyun Lee, Hyungyo Kim, Yongsuk Kwon, Jinhyun Kim, Jieun Lee, YeonGon Cho, Yongmin Tai, Jeonghyeon Cho, Hoyoung Song, Jung Ho Ahn, and Nam Sung Kim. An lpddr-based cxl-pnm platform for tco-efficient inference of transformer-based large language models. In *2024 IEEE International Symposium on High-Performance Computer Architecture (HPCA)*, pages 970–982, 2024. doi:10.1109/HPCA57654.2024.00078. URL https://doi.org/10.1109/HPCA57654.2024.00078.

[62] Md Rizwan Parvez, Wasi Ahmad, Saikat Chakraborty, Baishakhi Ray, and Kai-Wei Chang. Retrieval augmented code generation and summarization. In Marie-Francine Moens, Xuanjing Huang, Lucia Specia, and Scott Wen-tau Yih, editors, *Findings of the Association for Computational Linguistics: EMNLP 2021*, pages 2719–2734, Punta Cana, Dominican Republic, November 2021. Association for Computational Linguistics. doi:10.18653/v1/2021.findings-emnlp.232. URL https://aclanthology.org/2021.findings-emnlp.232.

[63] Dylan Patel. Apple M2 Die Shot and Architecture Analysis – Big Cost Increase And A15 Based IP. *SemiAnalysis*, June 2022. URL https://www.semianalysis.com/p/apple-m2-die-shot-and-architecture.

[64] N. Patel, A. Mamandipoor, M. Nouri, and M. Alian. Smartdimm: In-memory acceleration of upper layer protocols. In *2024 IEEE International Symposium on High-Performance Computer Architecture (HPCA)*, pages 312–329, Los Alamitos, CA, USA, mar 2024. IEEE Computer Society. doi:10.1109/HPCA57654.2024.00032. URL https://doi.ieeecomputersociety.org/10.1109/HPCA57654.2024.00032.

[65] Neel Patel, Amin Mamandipoor, Derrick Quinn, and Mohammad Alian. Xfm: Accelerated software-defined far memory. In *Proceedings of the 56th Annual IEEE/ACM International Symposium on Microarchitecture*, MICRO '23, page 769–783, New York, NY, USA, 2023. Association for Computing Machinery. ISBN 9798400703294. doi:10.1145/3613424.3623776. URL https://doi.org/10.1145/3613424.3623776.

[66] Pratyush Patel, Esha Choukse, Chaojie Zhang, Aashaka Shah, Inigo Goiri, Saeed Maleki, and Ricardo Bianchini. Splitwise: Efficient Generative LLM Inference Using Phase Splitting . In *2024 ACM/IEEE 51st Annual International Symposium on Computer Architecture (ISCA)*, pages 118–132, Los Alamitos, CA, USA, July 2024. IEEE Computer Society. doi:10.1109/ISCA59077.2024.00019. URL https://doi.ieeecomputersociety.org/10.1109/ISCA59077.2024.00019.

[67] Hao Peng, Ankur Parikh, Manaal Faruqui, Bhuwan Dhingra, and Dipanjan Das. Text generation with exemplar-based adaptive decoding. In Jill Burstein, Christy Doran, and Thamar Solorio, editors, *Proceedings of the 2019 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies, Volume 1 (Long and Short Papers)*, pages 2555–2565, Minneapolis, Minnesota, June 2019. Association for Computational Linguistics. doi:10.18653/v1/N19-1263. URL https://aclanthology.org/N19-1263.

[68] Fabio Petroni, Aleksandra Piktus, Angela Fan, Patrick Lewis, Majid Yazdani, Nicola De Cao, James Thorne, Yacine Jernite, Vladimir Karpukhin, Jean Maillard, Vassilis Plachouras, Tim Rocktäschel, and Sebastian Riedel. KILT: a benchmark for knowledge intensive language tasks. In Kristina Toutanova, Anna Rumshisky, Luke Zettlemoyer, Dilek Hakkani-Tur, Iz Beltagy, Steven Bethard, Ryan Cotterell, Tanmoy Chakraborty, and Yichao Zhou, editors, *Proceedings of the 2021 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies*, pages 2523–2544, Online, June 2021. Association for Computational Linguistics. doi:10.18653/v1/2021.naacl-main.200. URL https://aclanthology.org/2021.naacl-main.200.

[69] Yingqi Qu, Yuchen Ding, Jing Liu, Kai Liu, Ruiyang Ren, Wayne Xin Zhao, Daxiang Dong, Hua Wu, and Haifeng Wang. RocketQA: An optimized training approach to dense passage retrieval for open-domain question answering. In Kristina Toutanova, Anna Rumshisky, Luke Zettlemoyer, Dilek Hakkani-Tur, Iz Beltagy, Steven Bethard, Ryan Cotterell, Tanmoy Chakraborty, and Yichao Zhou, editors, *Proceedings of the 2021 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies*, pages 5835–5847, Online, June 2021. Association for Computational Linguistics. doi:10.18653/v1/2021.naacl-main.466. URL https://aclanthology.org/2021.naacl-main.466.

[70] Colin Raffel, Noam Shazeer, Adam Roberts, Katherine Lee, Sharan Narang, Michael Matena, Yanqi Zhou, Wei Li, and Peter J. Liu. Exploring the limits of transfer learning with a unified text-to-text transformer. *The Journal of Machine Learning Research*, 21(1), jan 2020. ISSN 1532-4435. URL https://doi.org/10.5555/3455716.3455856.

[71] Pranav Rajpurkar, Jian Zhang, Konstantin Lopyrev, and Percy Liang. SQuAD: 100,000+ questions for machine comprehension of text. In Jian Su, Kevin Duh, and Xavier Carreras, editors, *Proceedings of the 2016 Conference on Empirical Methods in Natural Language Processing*, pages 2383–2392, Austin, Texas, November 2016. Association for Computational Linguistics. doi:10.18653/v1/D16-1264. URL https://aclanthology.org/D16-1264.

[72] Rita Ramos, Desmond Elliott, and Bruno Martins. Retrieval-augmented image captioning. In Andreas Vlachos and Isabelle Augenstein, editors, *Proceedings of the 17th Conference of the European Chapter of the Association for Computational Linguistics*, pages 3666–3681, Dubrovnik, Croatia, May 2023. Association for Computational Linguistics. doi:10.18653/v1/2023.eacl-main.266. URL https://aclanthology.org/2023.eacl-main.266.

[73] Alireza Salemi and Hamed Zamani. Evaluating retrieval quality in retrieval-augmented generation. In *Proceedings of the 47th International ACM SIGIR Conference on Research and Development in Information Retrieval*, SIGIR '24, page 2395–2400, New York, NY, USA, 2024. Association for Computing Machinery. ISBN 9798400704314. doi:10.1145/3626772.3657957. URL https://doi.org/10.1145/3626772.3657957.

[74] Alireza Salemi and Hamed Zamani. Learning to rank for multiple retrieval-augmented models through iterative utility maximization, 2024. URL https://arxiv.org/abs/2410.09942.

[75] Alireza Salemi and Hamed Zamani. Comparing retrieval-augmentation and parameter-efficient fine-tuning for privacy-preserving personalization of large language models, 2024. URL https://arxiv.org/abs/2409.09510.

[76] Alireza Salemi and Hamed Zamani. Towards a search engine for machines: Unified ranking for multiple retrieval-augmented large language models. In *Proceedings of the 47th International ACM SIGIR Conference on Research and Development in Information Retrieval*, SIGIR '24, page 741–751, New York, NY, USA, 2024. Association for Computing Machinery. ISBN 9798400704314. doi:10.1145/3626772.3657733. URL https://doi.org/10.1145/3626772.3657733.

[77] Alireza Salemi, Juan Altmayer Pizzorno, and Hamed Zamani. A symmetric dual encoding dense retrieval framework for knowledge-intensive visual question answering. In *Proceedings of the 46th International ACM SIGIR Conference on Research and Development in Information Retrieval*, SIGIR '23, page 110–120, New York, NY, USA, 2023. Association for Computing Machinery. ISBN 9781450394086. doi:10.1145/3539618.3591629. URL https://doi.org/10.1145/3539618.3591629.

[78] Alireza Salemi, Mahta Rafiee, and Hamed Zamani. Pre-training multi-modal dense retrievers for outside-knowledge visual question answering. In *Proceedings of the 2023 ACM SIGIR International Conference on Theory of Information Retrieval*, ICTIR '23, page 169–176, New York, NY, USA, 2023. Association for Computing Machinery. ISBN 9798400700736. doi:10.1145/3578337.3605137. URL https://doi.org/10.1145/3578337.3605137.

[79] Alireza Salemi, Surya Kallumadi, and Hamed Zamani. Optimization methods for personalizing large language models through retrieval augmentation. In *Proceedings of the 47th International ACM SIGIR Conference on Research and Development in Information Retrieval*, SIGIR '24, page 752–762, New York, NY, USA, 2024. Association for Computing Machinery. ISBN 9798400704314. doi:10.1145/3626772.3657783. URL https://doi.org/10.1145/3626772.3657783.

[80] Alireza Salemi, Sheshera Mysore, Michael Bendersky, and Hamed Zamani. LaMP: When large language models meet personalization. In Lun-Wei Ku, Andre Martins, and Vivek Srikumar, editors, *Proceedings of the 62nd Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pages 7370–7392, Bangkok, Thailand, August 2024. Association for Computational Linguistics. doi:10.18653/v1/2024.acl-long.399. URL https://aclanthology.org/2024.acl-long.399.

[81] Satyabrata Sarangi and Bevan Baas. Deepscaletool: A tool for the accurate estimation of technology scaling in the deep-submicron era. In *2021 IEEE International Symposium on Circuits and Systems (ISCAS)*, pages 1–5, 2021. doi:10.1109/ISCAS51556.2021.9401196.

URL https://doi.org/10.1109/ISCAS51556.2021.9401196.

[82] Sara Sarto, Marcella Cornia, Lorenzo Baraldi, and Rita Cucchiara. Retrieval-augmented transformer for image captioning. In *Proceedings of the 19th International Conference on Content-Based Multimedia Indexing*, CBMI '22, page 1–7, New York, NY, USA, 2022. Association for Computing Machinery. ISBN 9781450397209. doi:10.1145/3549555.3549585. URL https://doi.org/10.1145/3549555.3549585.

[83] Henry N. Schuh, Arvind Krishnamurthy, David Culler, Henry M. Levy, Luigi Rizzo, Samira Khan, and Brent E. Stephens. Cc-nic: a cache-coherent interface to the nic. In *Proceedings of the 29th ACM International Conference on Architectural Support for Programming Languages and Operating Systems, Volume 1*, ASPLOS '24, page 52–68, New York, NY, USA, 2024. Association for Computing Machinery. ISBN 9798400703720. doi:10.1145/3617232.3624868. URL https://doi.org/10.1145/3617232.3624868.

[84] Haihao Shen, Hanwen Chang, Bo Dong, Yu Luo, and Hengyu Meng. Efficient llm inference on cpus. *arXiv preprint*, 2023. URL https://doi.org/10.48550/arXiv.2311.00502.

[85] Kurt Shuster, Spencer Poff, Moya Chen, Douwe Kiela, and Jason Weston. Retrieval augmentation reduces hallucination in conversation. In Marie-Francine Moens, Xuanjing Huang, Lucia Specia, and Scott Wen-tau Yih, editors, *Findings of the Association for Computational Linguistics: EMNLP 2021*, pages 3784–3803, Punta Cana, Dominican Republic, November 2021. Association for Computational Linguistics. doi:10.18653/v1/2021.findings-emnlp.320. URL https://aclanthology.org/2021.findings-emnlp.320.

[86] Joonseop Sim, Soohong Ahn, Taeyoung Ahn, Seungyong Lee, Myunghyun Rhee, Jooyoung Kim, Kwangsik Shin, Donguk Moon, Euiseok Kim, and Kyoung Park. Computational cxl-memory solution for accelerating memory-intensive applications. *IEEE Computer Architecture Letters*, 22(1):5–8, 2023. doi:10.1109/LCA.2022.3226482. URL https://doi.org/10.1109/LCA.2022.3226482.

[87] Shamane Siriwardhana, Rivindu Weerasekera, Elliott Wen, Tharindu Kaluarachchi, Rajib Rana, and Suranga Nanayakkara. Improving the Domain Adaptation of Retrieval Augmented Generation (RAG) Models for Open Domain Question Answering. *Transactions of the Association for Computational Linguistics*, 11:1–17, 01 2023. ISSN 2307-387X. doi:10.1162/tacl_a_00530. URL https://doi.org/10.1162/tacl_a_00530.

[88] Heidi Steen and Dan Wahlin. Retrieval augumented generation overview. *Microsoft Learn*, 2023. URL https://learn.microsoft.com/en-us/azure/search/retrieval-augmented-generation-overview.

[89] Jovan Stojkovic, Esha Choukse, Chaojie Zhang, Inigo Goiri, and Josep Torrellas. Towards greener llms: Bringing energy-efficiency to the forefront of llm inference. *arXiv preprint arXiv:2403.20306*, 2024. URL https://doi.org/10.48550/arXiv.2403.20306.

[90] Lisa T. Su, Samuel Naffziger, and Mark Papermaster. Multi-chip technologies to unleash computing performance gains over the next decade. In *2017 IEEE International Electron Devices Meeting (IEDM)*, pages 1.1.1–1.1.8, 2017. doi:10.1109/IEDM.2017.8268306. URL https://doi.org/10.1109/IEDM.2017.8268306.

[91] Yixuan Su, David Vandyke, Simon Baker, Yan Wang, and Nigel Collier. Keep the primary, rewrite the secondary: A two-stage approach for paraphrase generation. In Chengqing Zong, Fei Xia, Wenjie Li, and Roberto Navigli, editors, *Findings of the Association for Computational Linguistics: ACL-IJCNLP 2021*, pages 560–569, Online, August 2021. Association for Computational Linguistics. doi:10.18653/v1/2021.findings-acl.50. URL https://aclanthology.org/2021.findings-acl.50.

[92] Gemini Team. Gemini: A family of highly capable multimodal models. *arXiv*, 2024. URL https://doi.org/10.48550/arXiv.2312.11805.

[93] David Thulke, Nico Daheim, Christian Dugast, and Hermann Ney. Efficient retrieval augmented generation from unstructured knowledge for task-oriented dialog. *arXiv preprint arXiv:2102.04643*, 2021. URL https://doi.org/10.48550/arXiv.2102.04643.

[94] Zhiliang Tian, Wei Bi, Xiaopeng Li, and Nevin L. Zhang. Learning to abstract for memory-augmented conversational response generation. In Anna Korhonen, David Traum, and Lluís Màrquez, editors, *Proceedings of the 57th Annual Meeting of the Association for Computational Linguistics*, pages 3816–3825, Florence, Italy, July 2019. Association for Computational Linguistics. doi:10.18653/v1/P19-1371. URL https://aclanthology.org/P19-1371.

[95] Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N Gomez, Ł ukasz Kaiser, and Illia Polosukhin. Attention is all you need. In I. Guyon, U. Von Luxburg, S. Bengio, H. Wallach, R. Fergus, S. Vishwanathan, and R. Garnett, editors, *Advances in Neural Information Processing Systems*, volume 30. Curran Associates, Inc., 2017. URL https://proceedings.neurips.cc/paper_files/paper/2017/file/3f5ee243547dee91fbd053c1c4a845aa-Paper.pdf.

[96] Mengzhao Wang, Xiaoliang Xu, Qiang Yue, and Yuxiang Wang. A comprehensive survey and experimental comparison of graph-based approximate nearest neighbor search. *Proceedings of the VLDB Endowment*, 14(11):1964–1978, jul 2021. ISSN 2150-8097. doi:10.14778/3476249.3476255. URL https://doi.org/10.14778/3476249.3476255.

[97] Yitu Wang, Shiyu Li, Qilin Zheng, Linghao Song, Zongwang Li, Andrew Chang, Hai "Helen" Li, and Yiran Chen. Ndsearch: Accelerating graph-traversal-based approximate nearest neighbor search through near data processing. In *Proceedings of the 39th Annual International Symposium on Computer Architecture*, 2024. URL https://doi.org/10.48550/arXiv.2312.03141.

[98] Zelin Wang, Ping Gong, Yibo Zhang, Jihao Gu, and Xuanyuan Yang. Retrieval-augmented knowledge-intensive dialogue. In Fei Liu, Nan Duan, Qingting Xu, and Yu Hong, editors, *Natural Language Processing and Chinese Computing*, pages 16–28, Cham, 2023. Springer Nature Switzerland. ISBN 978-3-031-44693-1. URL https://doi.org/10.48550/arXiv.2005.11401.

[99] Jason Weston, Emily Dinan, and Alexander Miller. Retrieve and refine: Improved sequence generation models for dialogue. In Aleksandr Chuklin, Jeff Dalton, Julia Kiseleva, Alexey Borisov, and Mikhail Burtsev, editors, *Proceedings of the 2018 EMNLP Workshop SCAI: The 2nd International Workshop on Search-Oriented Conversational AI*, pages 87–92, Brussels, Belgium, October 2018. Association for Computational Linguistics. doi:10.18653/v1/W18-5713. URL https://aclanthology.org/W18-5713.

[100] Yu Wu, Furu Wei, Shaohan Huang, Yunli Wang, Zhoujun Li, and Ming Zhou. Response generation by context-aware prototype editing. *Proceedings of the AAAI Conference on Artificial Intelligence*, 33 (01):7281–7288, Jul. 2019. doi:10.1609/aaai.v33i01.33017281. URL https://ojs.aaai.org/index.php/AAAI/article/view/4714.

[101] Guangxuan Xiao, Ji Lin, Mickael Seznec, Hao Wu, Julien Demouth, and Song Han. Smoothquant: Accurate and efficient post-training quantization for large language models. In *International Conference on Machine Learning*, pages 38087–38099. PMLR, 2023. URL https://doi.org/10.5555/3618408.3619993.

[102] Guangxuan Xiao, Yuandong Tian, Beidi Chen, Song Han, and Mike Lewis. Efficient streaming language models with attention sinks. In *The Twelfth International Conference on Learning Representations*, 2024. URL https://openreview.net/forum?id=NG7sS51zVF.

[103] Jitao Xu, Josep Crego, and Jean Senellart. Boosting neural machine translation with similar translations. In Dan Jurafsky, Joyce Chai, Natalie Schluter, and Joel Tetreault, editors, *Proceedings of the 58th Annual Meeting of the Association for Computational Linguistics*, pages 1580–1590, Online, July 2020. Association for Computational Linguistics. doi:10.18653/v1/2020.acl-main.144. URL https://aclanthology.org/2020.acl-main.144.

[104] Shi-Qi Yan, Jia-Chen Gu, Yun Zhu, and Zhen-Hua Ling. Corrective retrieval augmented generation. *arXiv*, 2024.

doi:10.48550/arXiv.2401.15884.

[105] Nan Yang, Tao Ge, Liang Wang, Binxing Jiao, Daxin Jiang, Linjun Yang, Rangan Majumder, and Furu Wei. Inference with reference: Lossless acceleration of large language models. *arXiv preprint arXiv:2304.04487*, 2023. URL https://doi.org/10.48550/arXiv.2304.04487.

[106] Yifan Yuan, Jinghan Huang, Yan Sun, Tianchen Wang, Jacob Nelson, Dan R. K. Ports, Yipeng Wang, Ren Wang, Charlie Tai, and Nam Sung Kim. Rambda: Rdma-driven acceleration framework for memory-intensive µs-scale datacenter applications. In *2023 IEEE International Symposium on High-Performance Computer Architecture (HPCA)*, pages 499–515, 2023. doi:10.1109/HPCA56546.2023.10071127. URL https://doi.org/10.1109/HPCA56546.2023.10071127.

[107] Hamed Zamani and Michael Bendersky. Stochastic rag: End-to-end retrieval-augmented generation through expected utility maximization. In *Proceedings of the 47th International ACM SIGIR Conference on Research and Development in Information Retrieval*, SIGIR '24, page 2641–2646, New York, NY, USA, 2024. Association for Computing Machinery. ISBN 9798400704314. doi:10.1145/3626772.3657923. URL https://doi.org/10.1145/3626772.3657923.

[108] Hamed Zamani, Fernando Diaz, Mostafa Dehghani, Donald Metzler, and Michael Bendersky. Retrieval-enhanced machine learning. In *Proceedings of the 45th International ACM SIGIR Conference on Research and Development in Information Retrieval*, SIGIR '22, page 2875–2886, New York, NY, USA, 2022. Association for Computing Machinery. ISBN 9781450387323. doi:10.1145/3477495.3531722. URL https://doi.org/10.1145/3477495.3531722.

[109] Jingyi Zhang, Masao Utiyama, Eiichro Sumita, Graham Neubig, and Satoshi Nakamura. Guiding neural machine translation with retrieved translation pieces. In Marilyn Walker, Heng Ji, and Amanda Stent, editors, *Proceedings of the 2018 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies, Volume 1 (Long Papers)*, pages 1325–1335, New Orleans, Louisiana, June 2018. Association for Computational Linguistics. doi:10.18653/v1/N18-1120. URL https://aclanthology.org/N18-1120.

[110] Yunan Zhang, Shige Liu, and Jianguo Wang. Are there fundamental limitations in supporting vector data management in relational databases? a case study of postgresql. *Preprint*. URL https://www.cs.purdue.edu/homes/csjgwang/pubs/ICDE24_VecDB.pdf. Accepted for publication in Proceedings of the International Conference on Data Engineering (ICDE).

[111] Zhe Zhou, Cong Li, Fan Yang, and Guangyu Sun. DIMM-Link: Enabling Efficient Inter-DIMM Communication for Near-Memory Processing. In *2023 IEEE International Symposium on High-Performance Computer Architecture (HPCA)*, pages 302–316, February 2023. doi:10.1109/HPCA56546.2023.10071005. URL https://doi.org/10.1109/HPCA56546.2023.10071005. ISSN: 2378-203X.

[112] Yun Zhu, Jia-Chen Gu, Caitlin Sikora, Ho Ko, Yinxiao Liu, Chu-Cheng Lin, Lei Shu, Liangchen Luo, Lei Meng, Bang Liu, and Jindong Chen. Accelerating inference of retrieval-augmented generation via sparse context selection. *arXiv*, 2024. doi:10.48550/arXiv.2405.16178.