Accelerating Retrieval-Augmented Generation

Derrick Quinn, Mohammad Nouri, Neel Patel,
Alireza Salimi[1], Sukhan Lee[2], Hamed Zamani[1], and Mohammad Alian

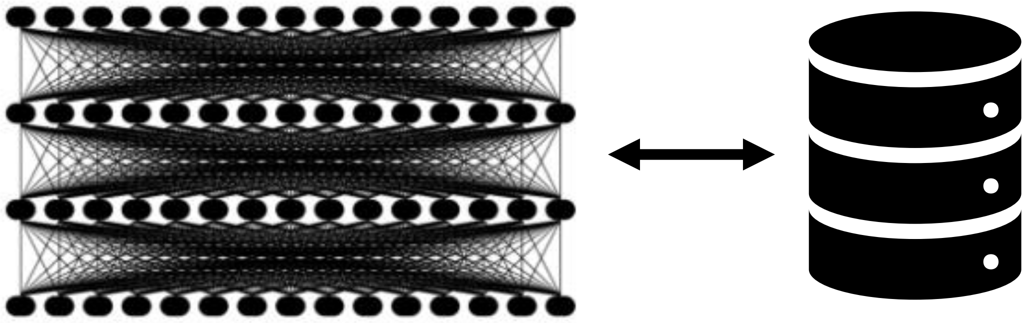
Cornell, UMass Amherst[1], Samsung Electronics[2]

CornellEngineering
Electrical and Computer Engineering

CSL

Retrieval Augmented Generation (RAG)

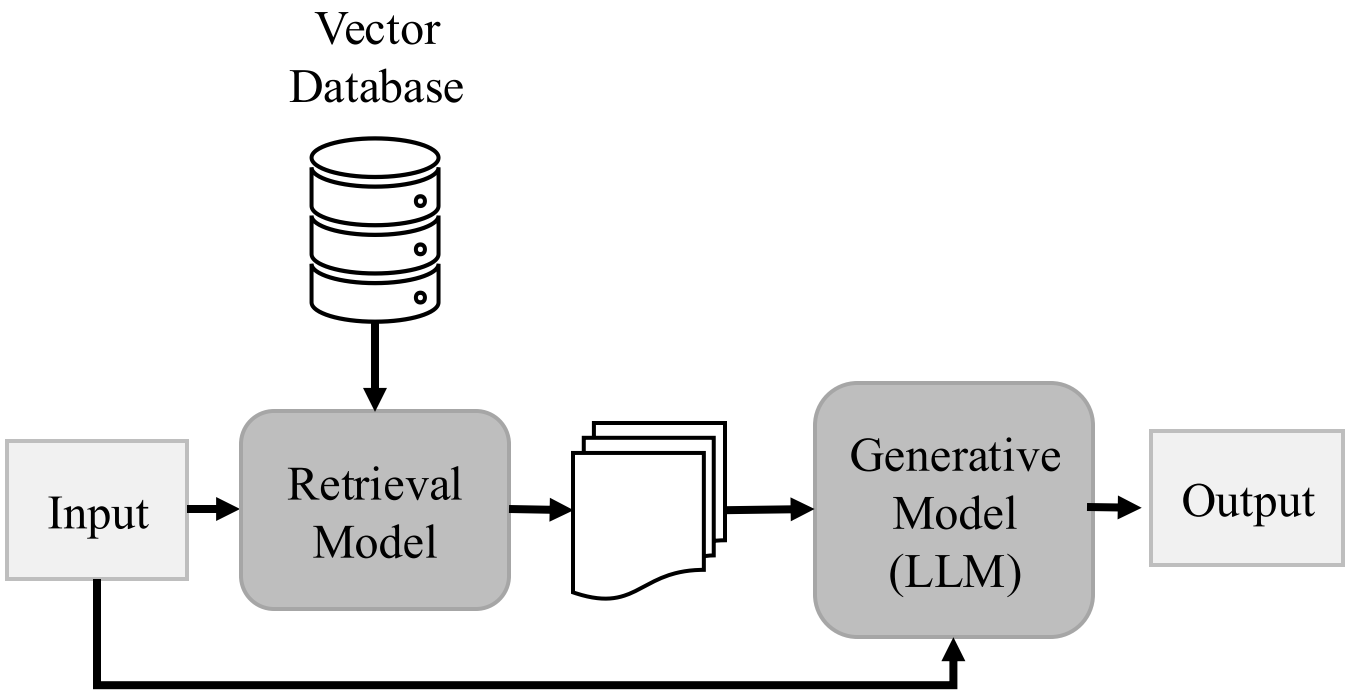
- Majority of LLM-powered applications use RAG
- Access to fresh data is a must have!
  - Updated knowledge, factual grounding



Pre-trained LLM Database

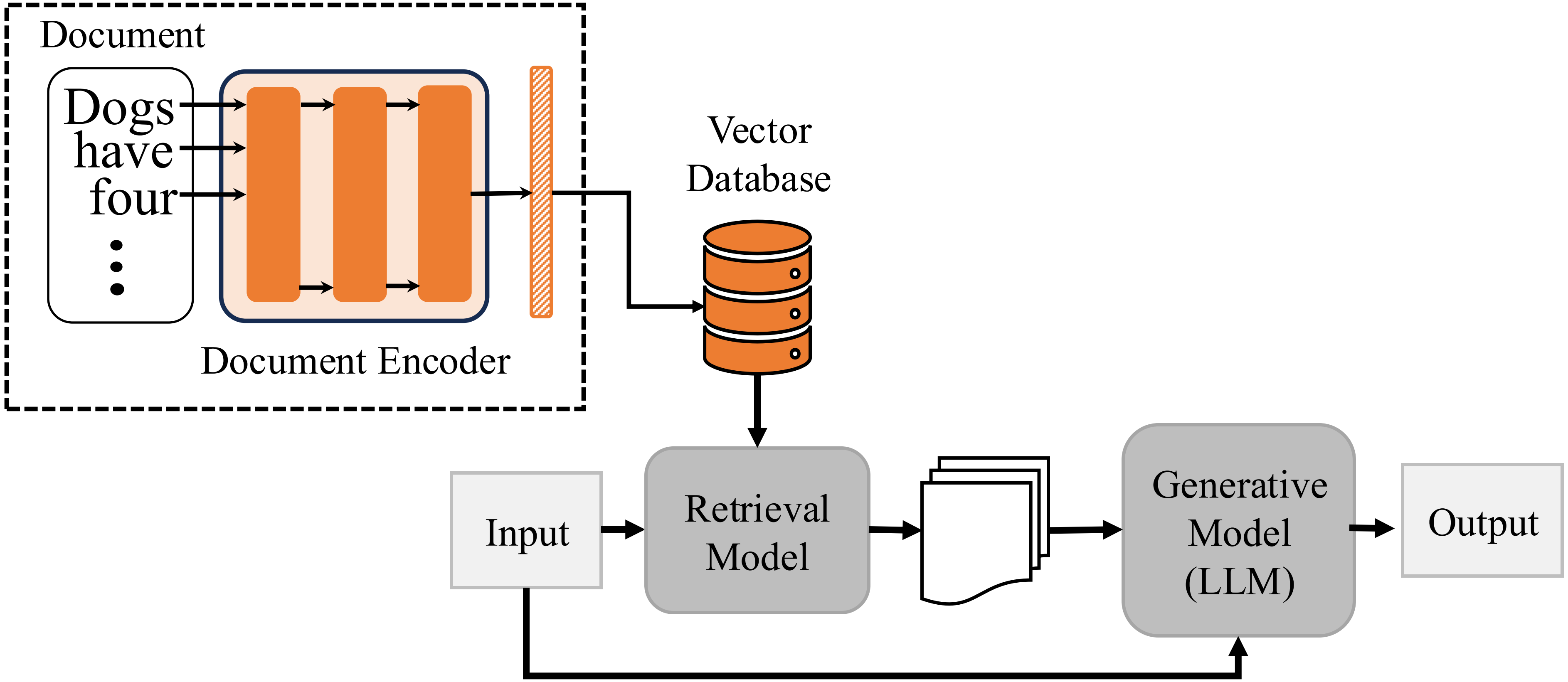
Image generated by ChatGPT!

RAG Pipeline Overview



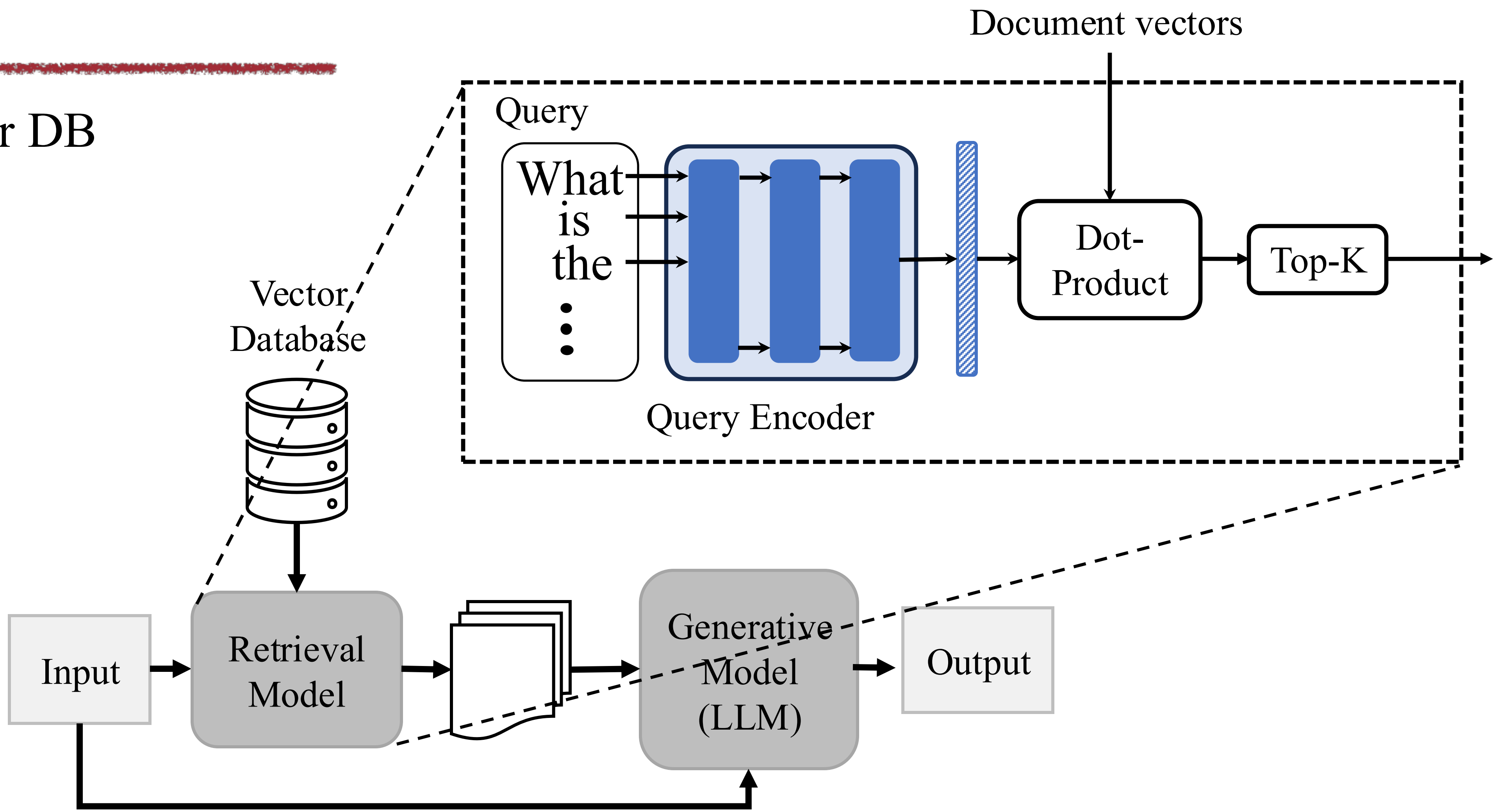
RAG Pipeline Overview

Offline: Populate Vector DB



RAG Pipeline Overview

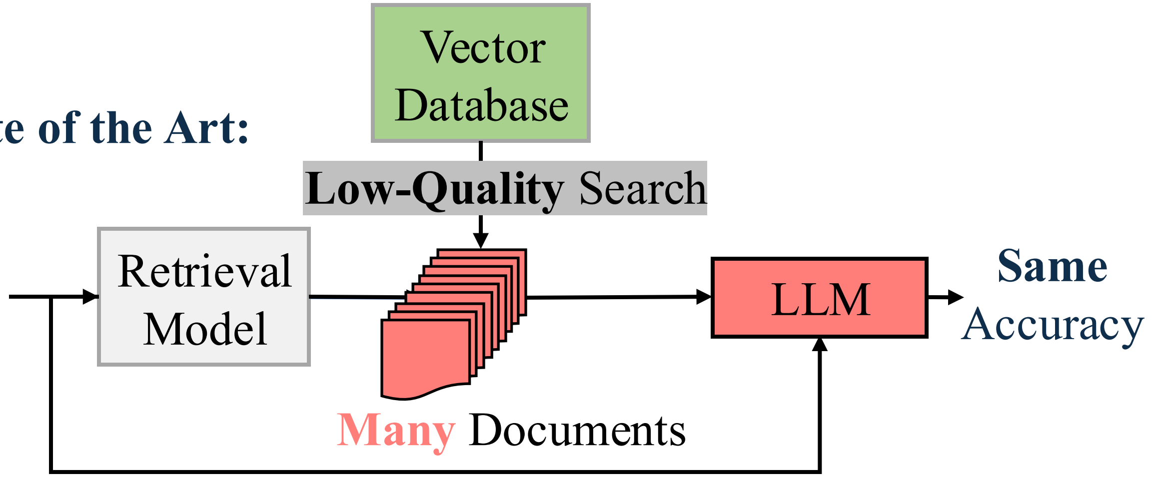
Online: Search Vector DB

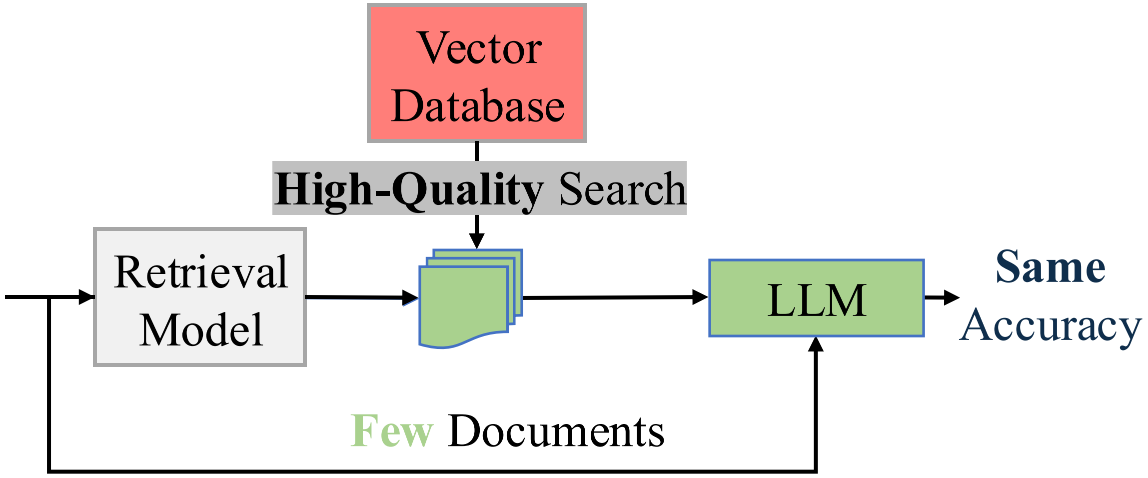


Insight: Interplay of Retrieval and Generation

	State of the Art	This Work
Accuracy	+++	+++
Retrieval Speed	+++	++
Generation Speed	--	+
Overall Speed	+	+++

State of the Art:





This Work (Near-Memory Acceleration of Exact Search)

Insight: Interplay of Retrieval and Generation

Vector Database

Our contributions:

1. We showcase the system-level interactions of retrieval quality
2. We leverage algorithm-hardware co-design to build a high-quality retrieval accelerator, accelerating RAG
3. We showcase the utility of CXL for the interface of high-capacity accelerators

Accuracy

Retrieval Speed

Generation Speed

Overall Speed

\+ +++

LM → **Same** Accuracy

LM → **Same** Accuracy

Few Documents

This Work (Near-Memory Acceleration of Exact Search)

Content

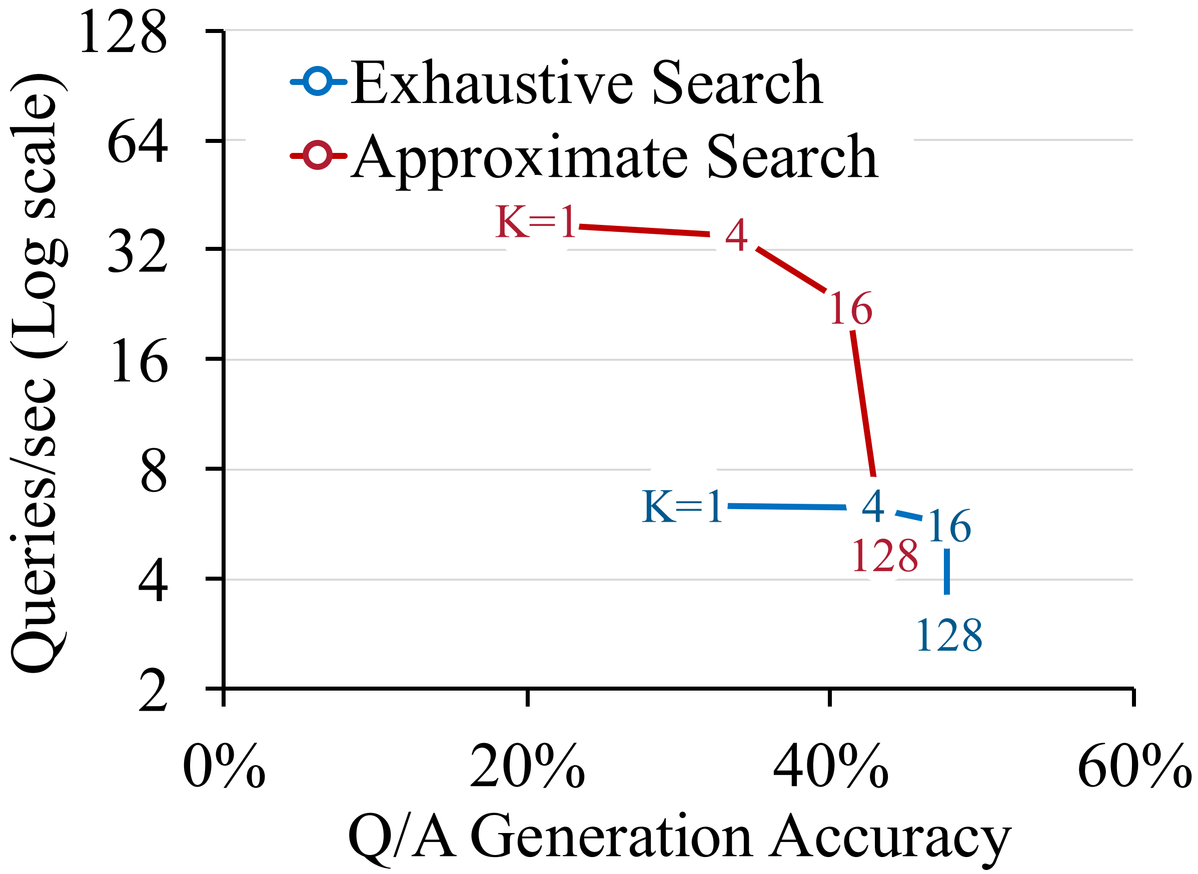
- Background
- Profiling RAG Applications
  - A Case for Near-Memory Exhaustive Search Acceleration
- Introducing Intelligent Knowledge Store (IKS)
- Evaluating IKS in a RAG pipeline

Profiling RAG Applications

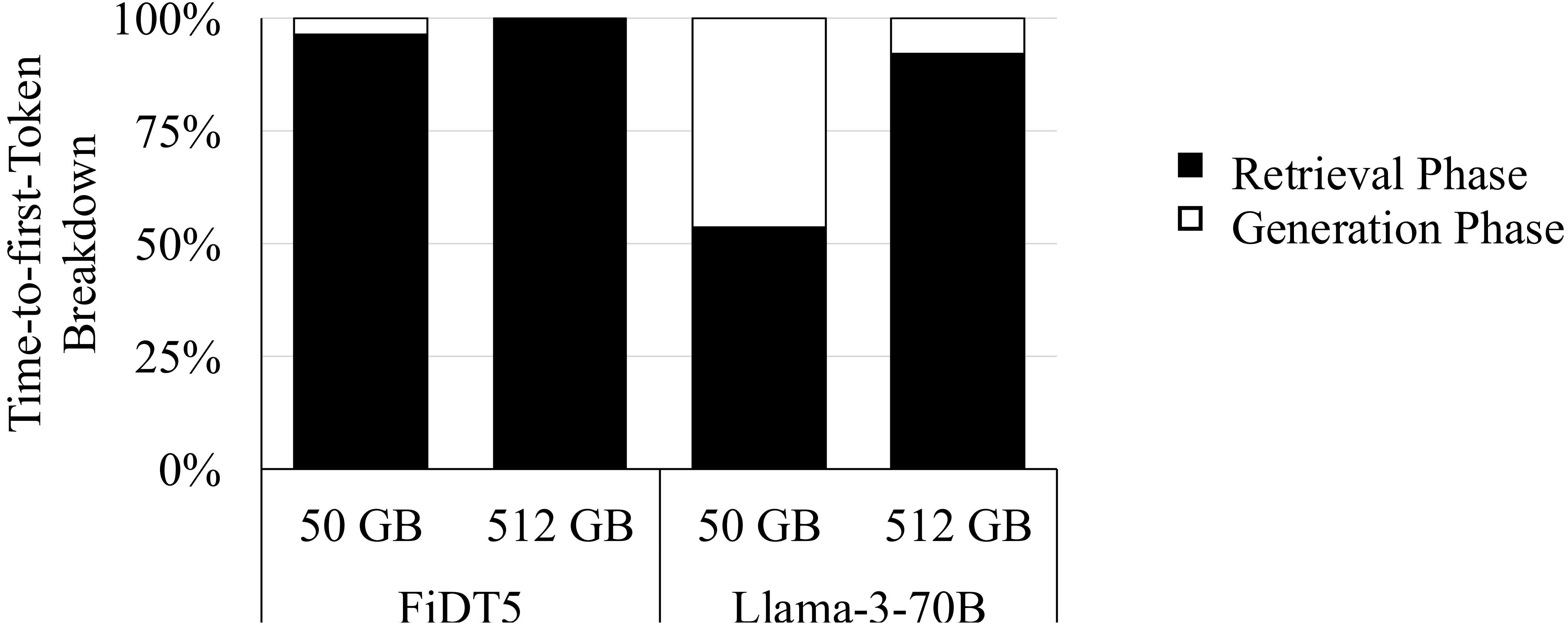
Evaluating RAG Applications: Methodology

- Question answering RAG applications
  - LLM: FiDT5 (T5 Fusion in Decoder), Llama-3-8B, Llama-3-70B
  - Retrieval model: BERT uncased to generate embeddings
  - 50-512GB vector database
- Meta FAISS library for similarity search
  - **Approximate** Nearest Neighbor **Search**: HNSW
  - **Exact/Exhaustive** Nearest Neighbor **Search**: Search the entire vector DB
- Experimental setup
  - Retrieval on CPU (Intel Xeon 4^{th} gen 4416+)
  - Generation on GPU (H100 80GB)
  - Cycle-approximate simulator for IKS (Ours)

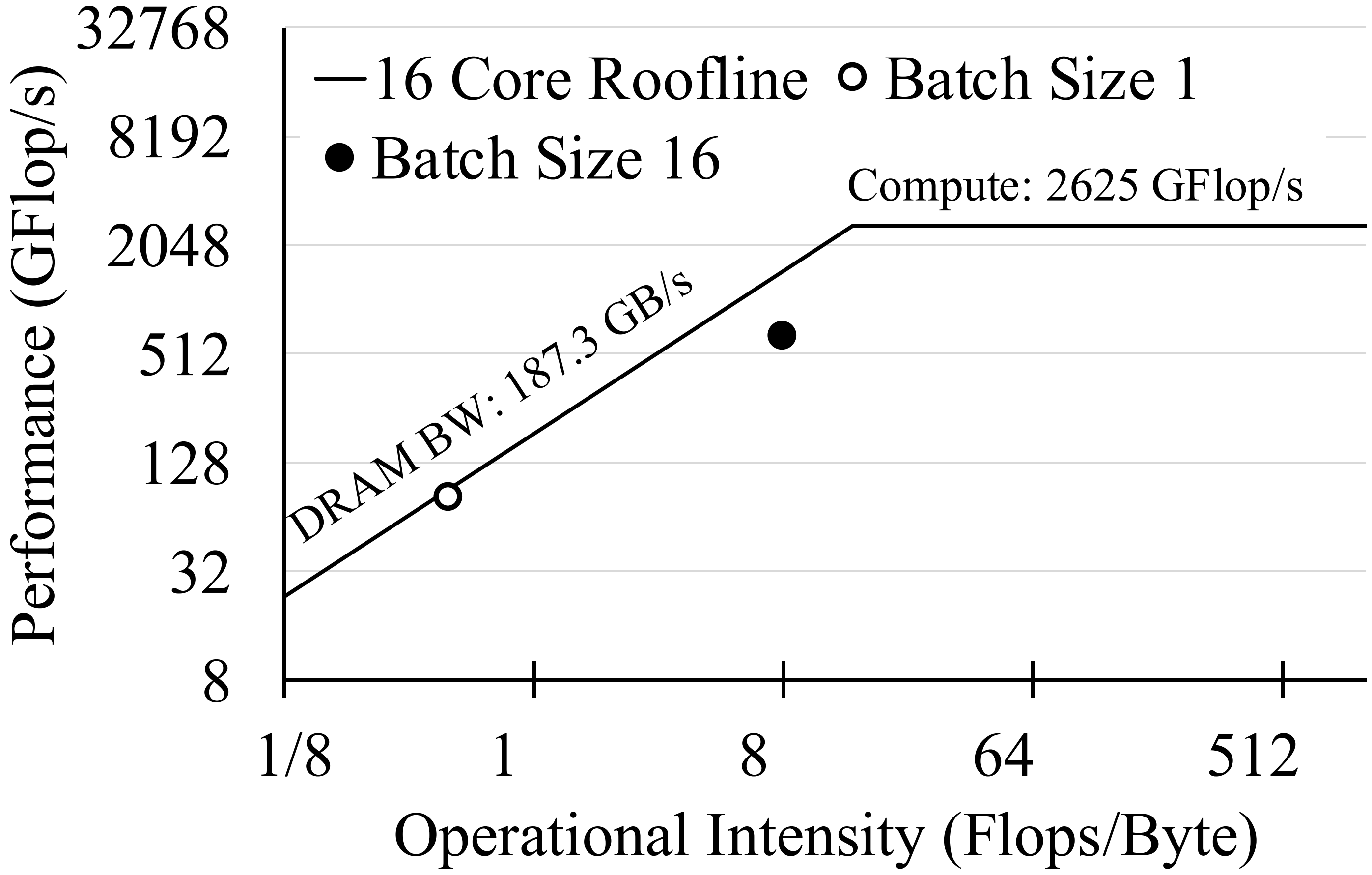
Observation 1: Trade-off Space



Observation 2: Exhaustive Search Bottlenecks RAG



Observation 3: Exhaustive Search is Memory-Bound



Case for Near-Memory Exhaustive Search Acceleration

Near-Memory Exhaustive Search Acceleration is Justified

Observations:

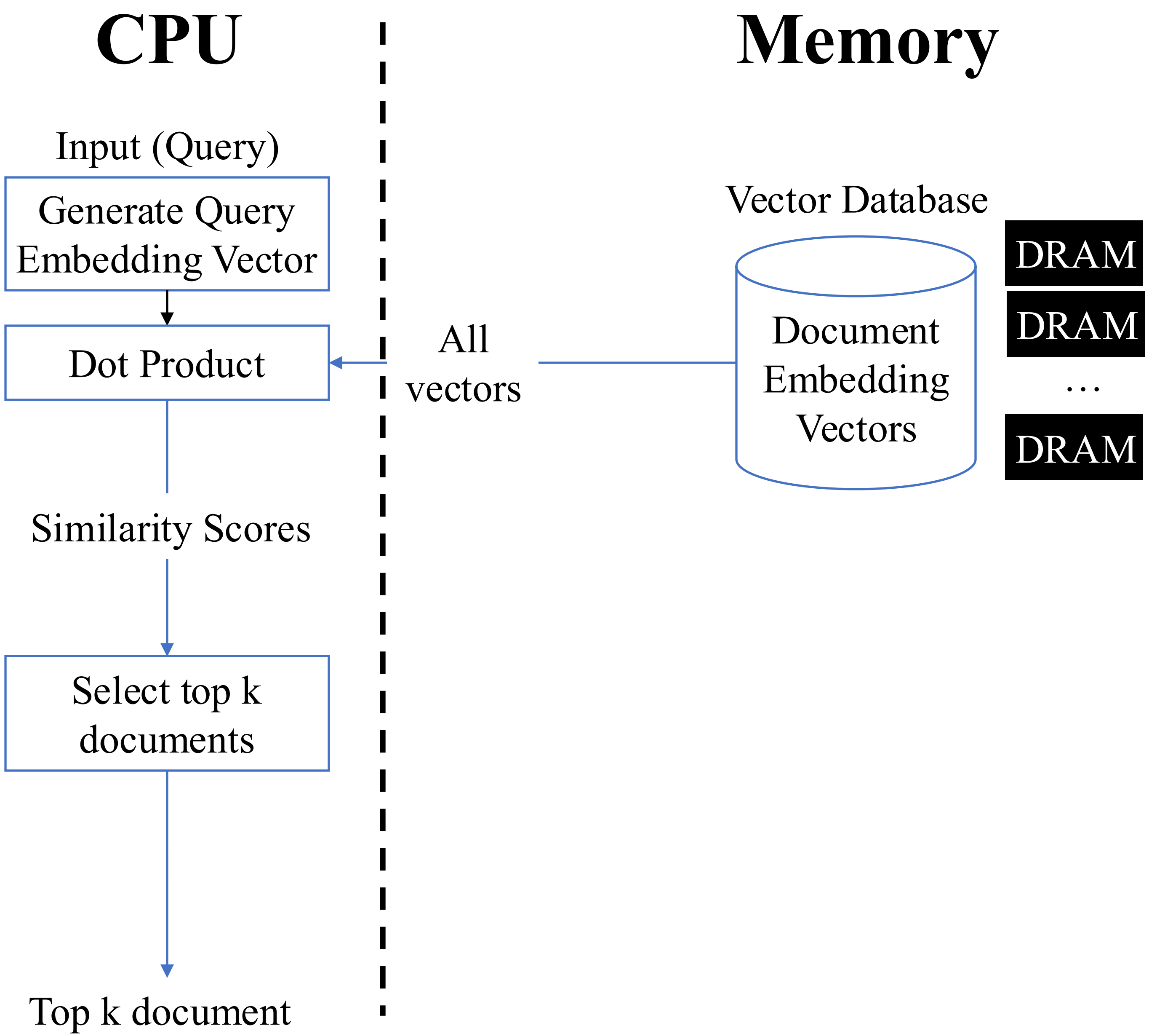
1. Exhaustive search provides useful accuracy for RAG
2. Exhaustive search can dominate time-to-first-token
3. Exhaustive search is memory-bound

- Therefore, we apply algorithm-hardware co-design:

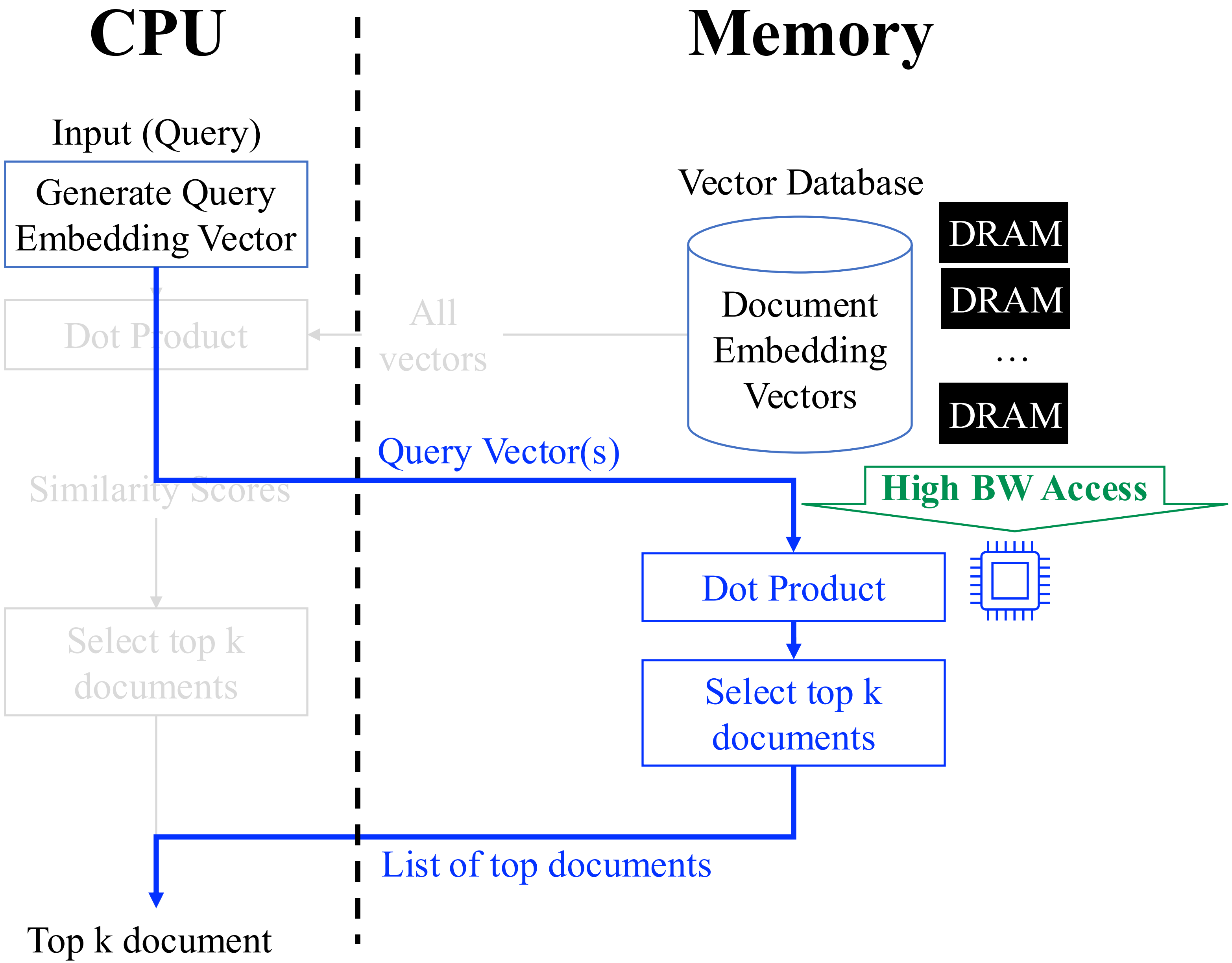
RAG Motivates Near-Memory Acceleration of Exhaustive Search

Intelligent Knowledge Store (IKS)

Baseline: CPU Retrieval

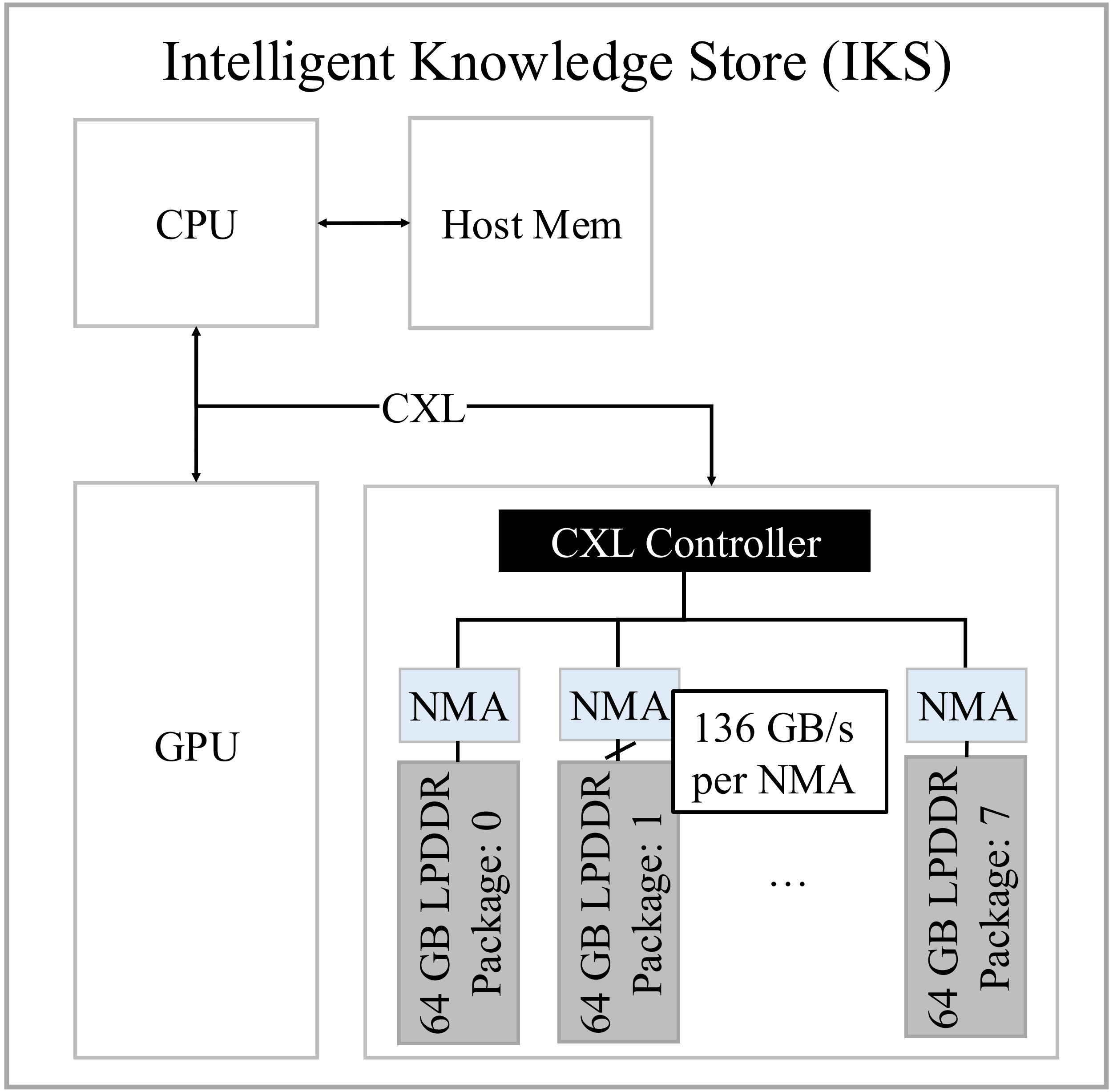


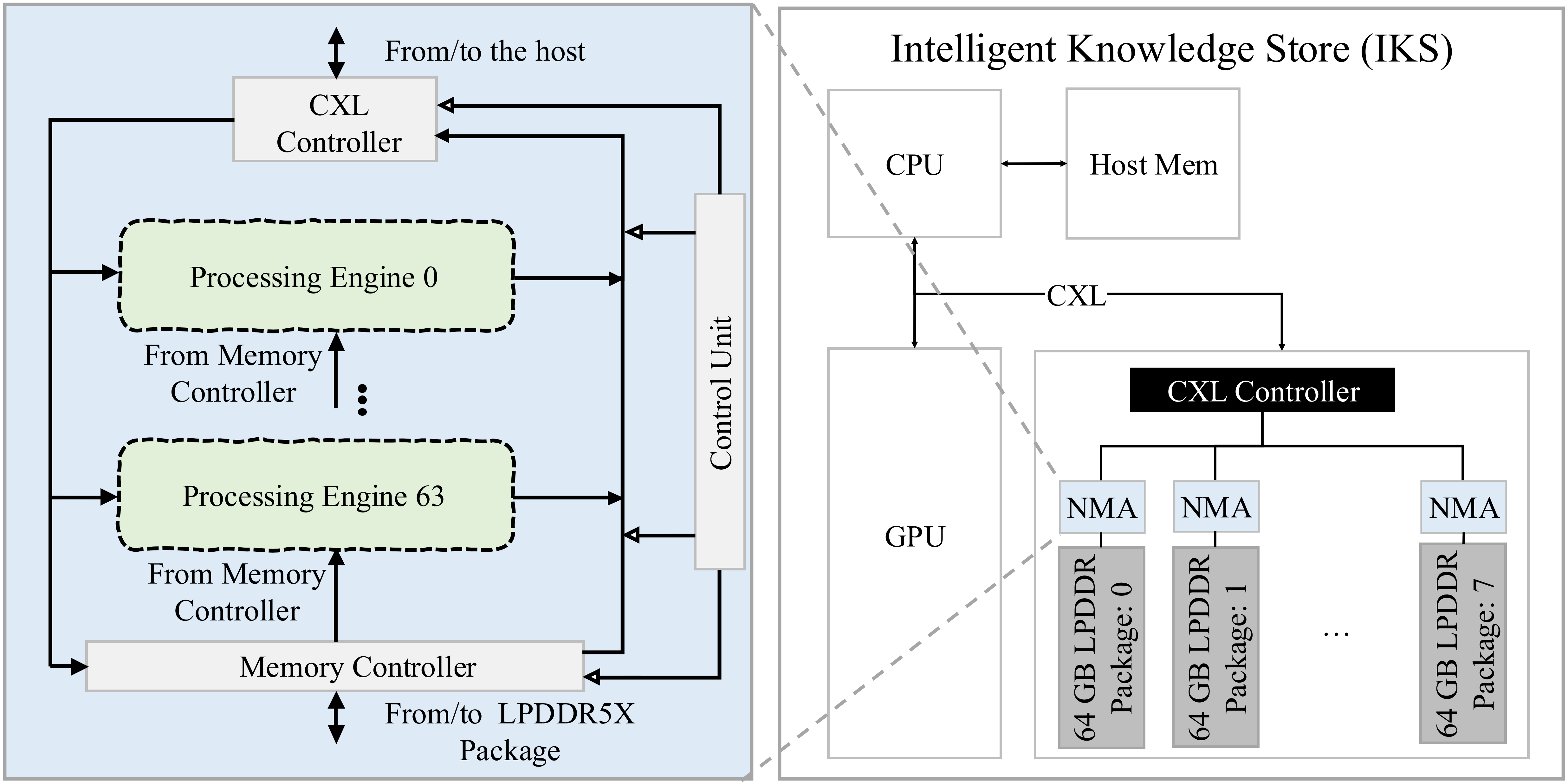
This work:
CPU + IKS
Retrieval

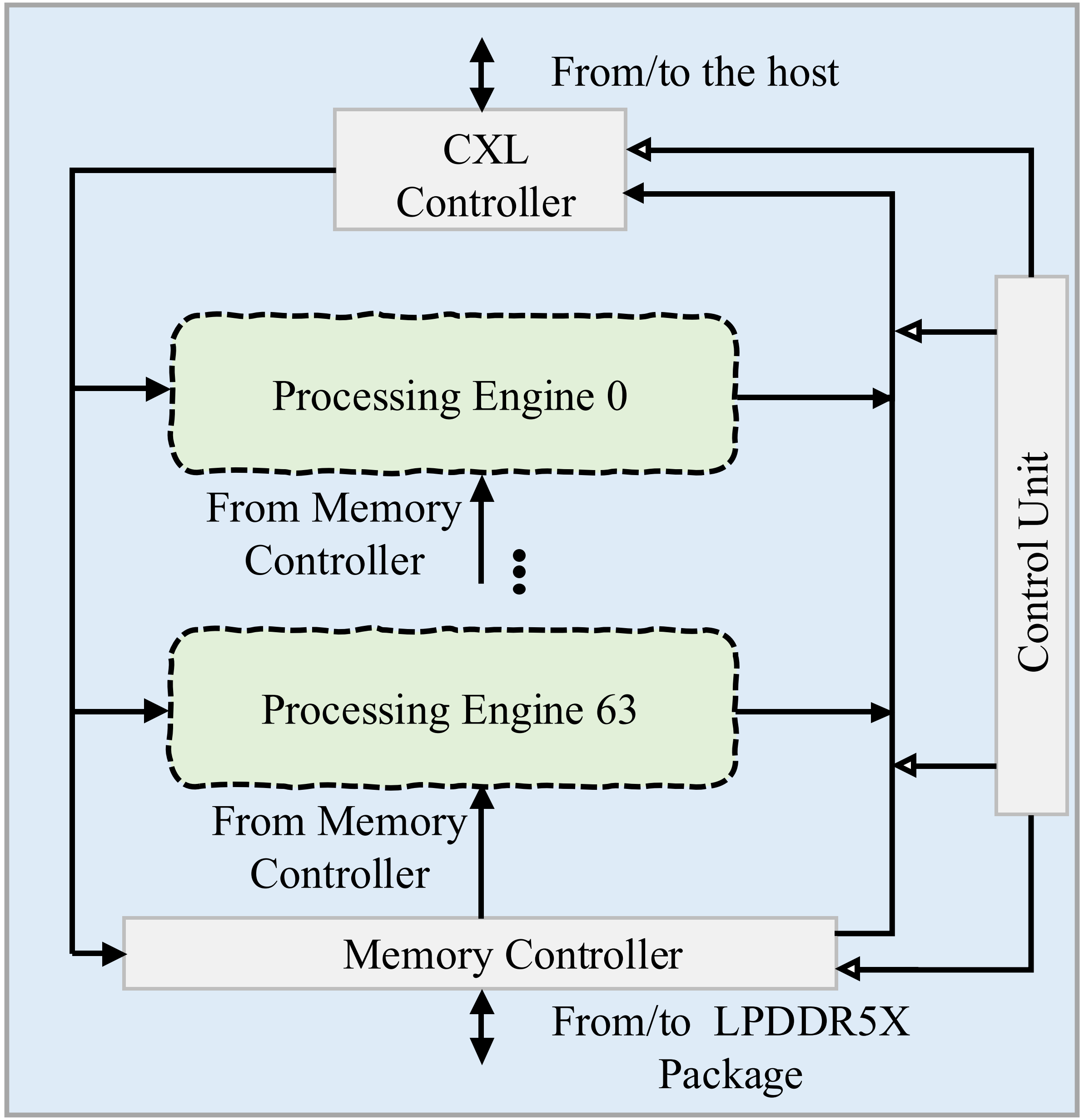


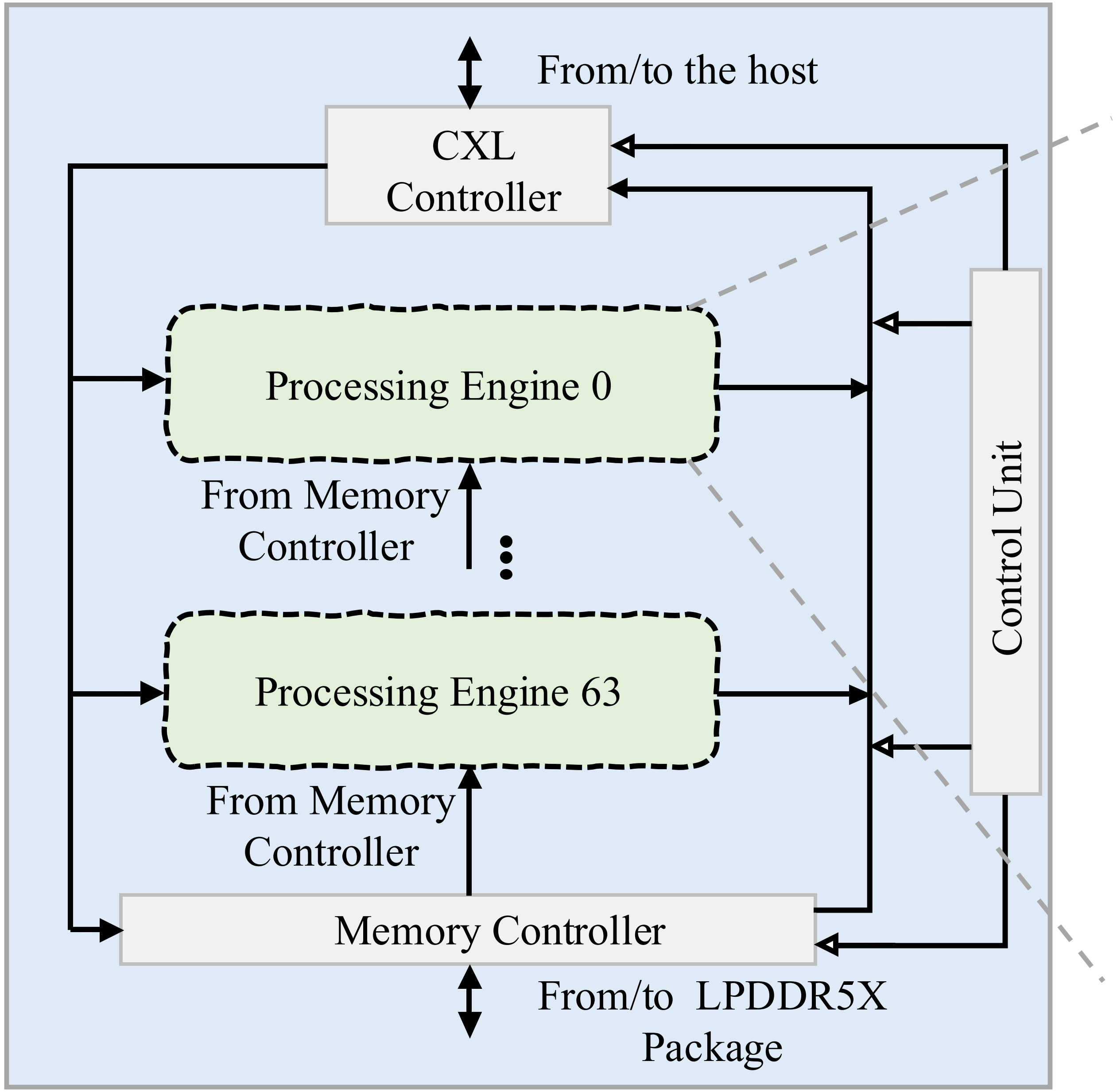
IKS Overview

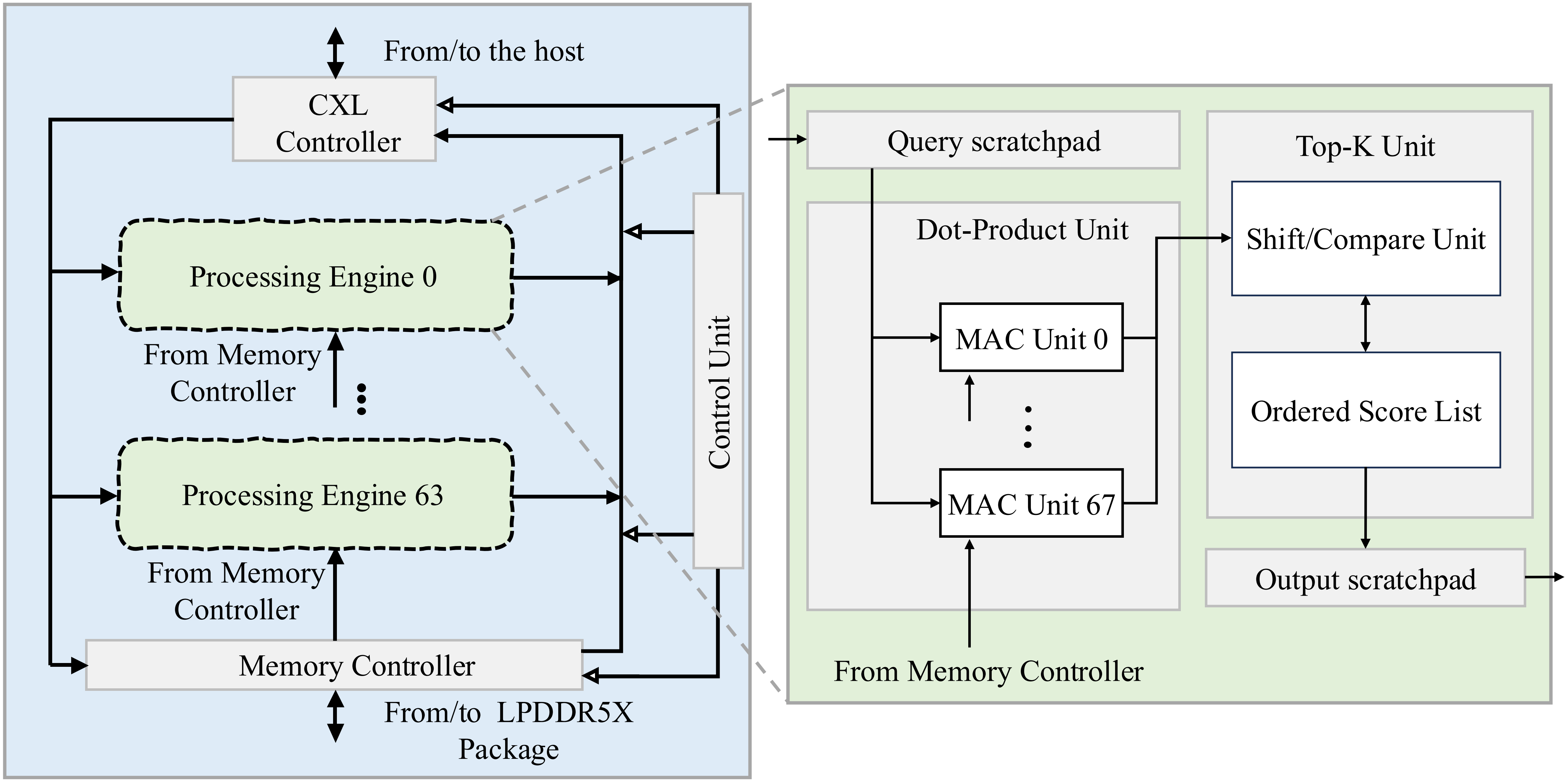
- IKS provisioned with:
  - 8x Near-Memory Accelerators (NMA)
  - 512 GB total capacity
  - 1.1 TB/sec internal bandwidth
- IKS supports:
  - Usermode polling
  - Multi-tenancy with host applications
  - Batch Size up to 64




From/to the host
CXL Controller
Processing Engine 0
From Memory Controller
Processing Engine 63
From Memory Controller
Memory Controller
From/to LPDDR5X Package
Control Unit
Intelligent Knowledge Store (IKS)
CPU
Host Mem
CXL
GPU
CXL Controller
NMA
NMA
NMA
64 GB LPDDR Package: 0
64 GB LPDDR Package: 1
…
64 GB LPDDR Package: 7


From/to the host
CXL Controller
Processing Engine 0
From Memory Controller
Processing Engine 63
From Memory Controller
Control Unit
Memory Controller
From/to LPDDR5X Package

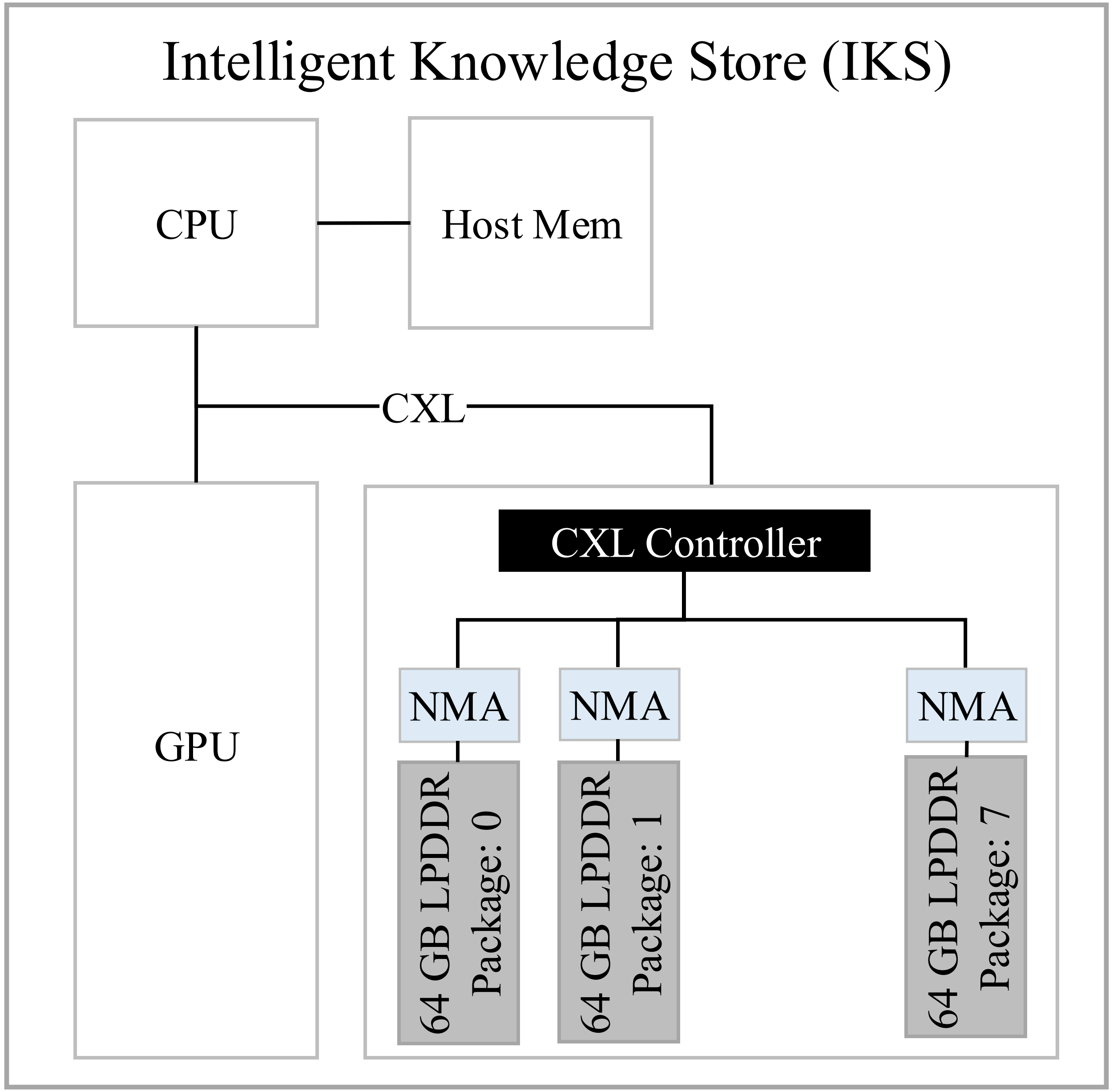

From/to the host
CXL
Controller
Processing Engine 0
From Memory
Controller
Processing Engine 63
From Memory
Controller
Control Unit
Memory Controller
From/to LPDDR5X
Package



From/to the host
CXL Controller
Processing Engine 0
From Memory Controller
Processing Engine 63
From Memory Controller
Memory Controller
From/to LPDDR5X Package
Control Unit
Query scratchpad
Dot-Product Unit
MAC Unit 0
MAC Unit 67
From Memory Controller
Top-K Unit
Shift/Compare Unit
Ordered Score List
Output scratchpad

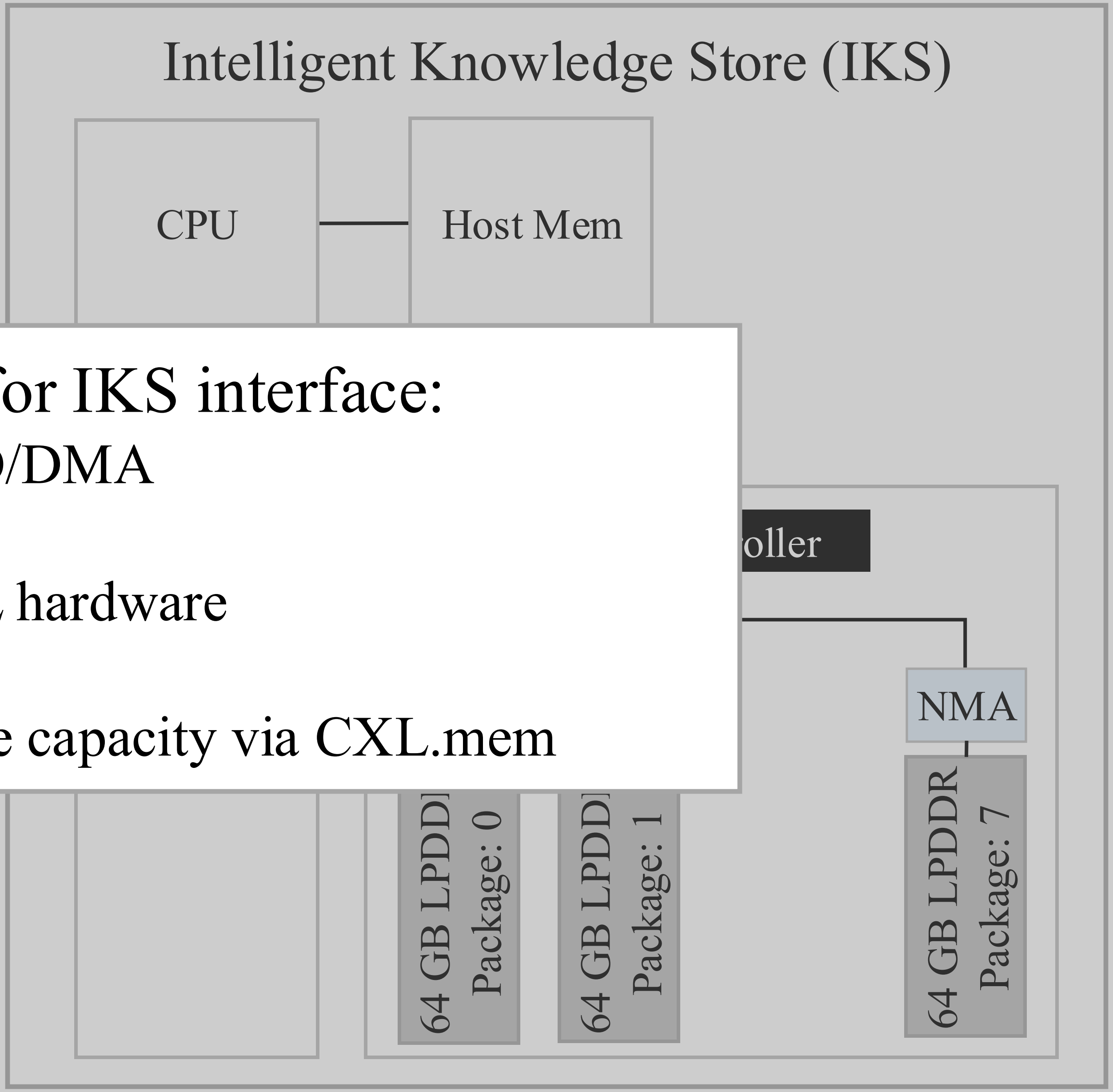
Leveraging CXL to implement IKS

- IKS *collaborates* with CPU!
- Option 1: DMA
  - High initial overhead
- Option 2: PIO (CXL.io)
  - Low bandwidth
- Our approach: Use CXL.cache
  - Build request in cache: high BW!
  - No host mem hop: low latency!



Leveraging CXL to implement IKS

- IKS *collaborates* with CPU!
- Option 1
  - High
- Option 2
  - Low b
- Our app
  - Build request in cache: high BW!
  - No host mem hop: low latency!

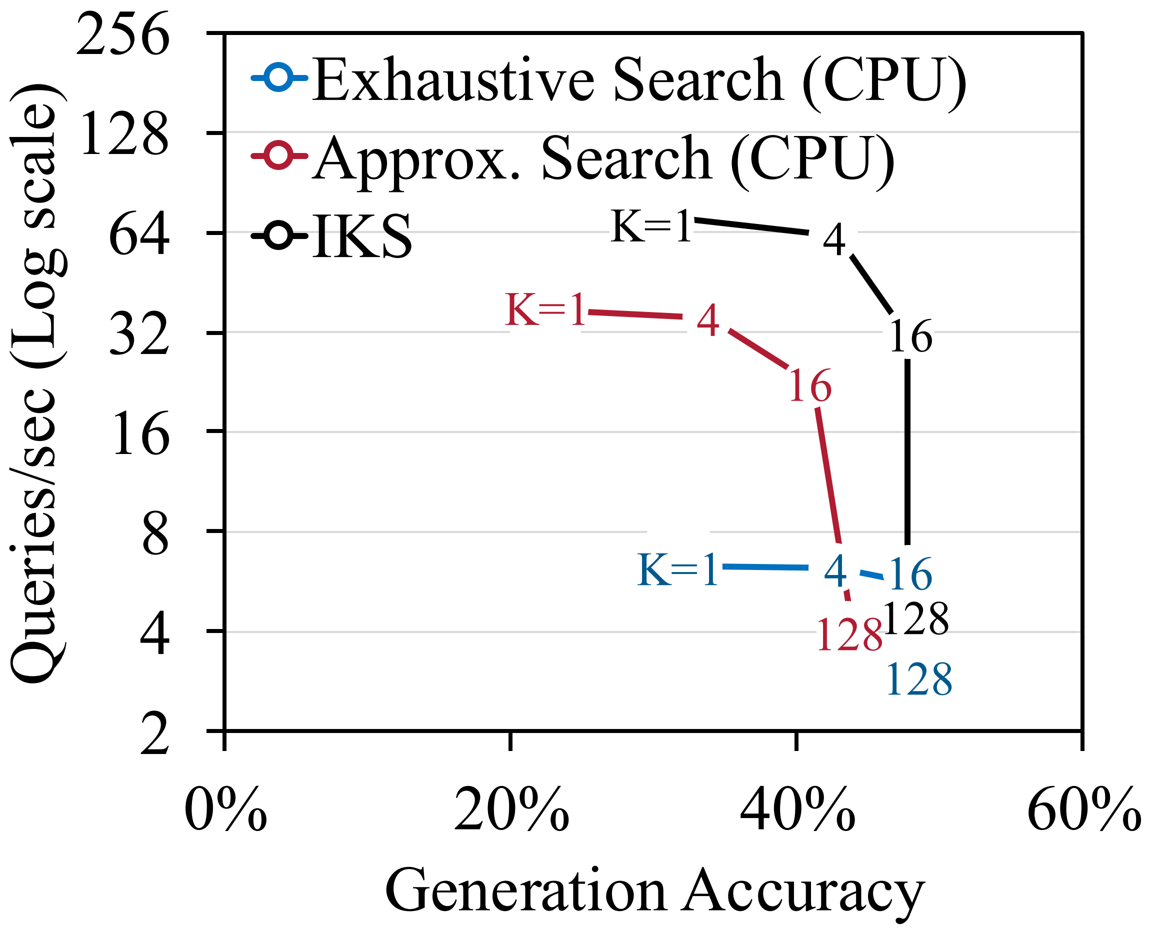


CXL.mem/cache for IKS interface:

- Improved performance vs. PIO/DMA
- Uses existing commodity CXL hardware
- Enables disaggregation of huge capacity via CXL.mem

Evaluating IKS

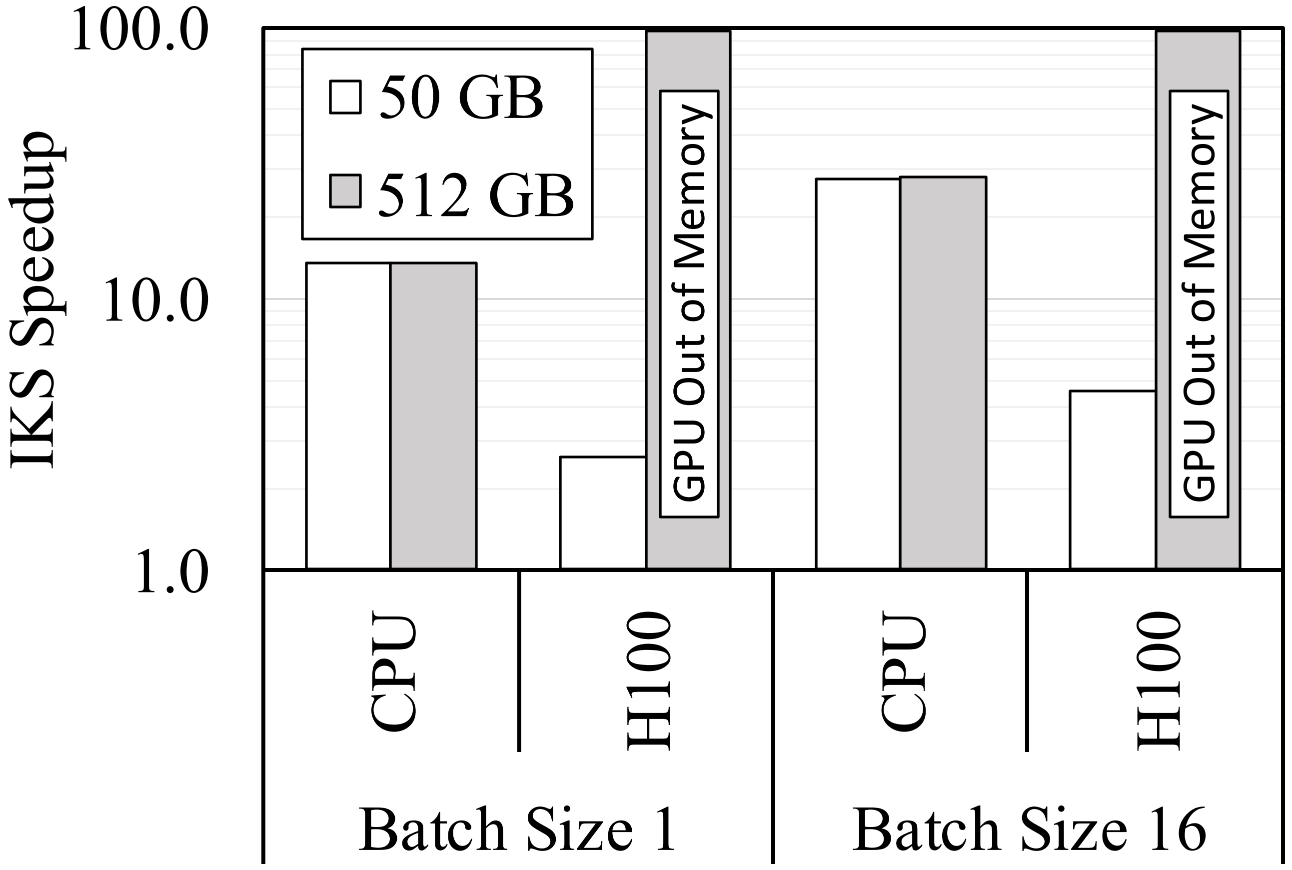
IKS Accelerates RAG



Effectiveness of IKS Retrieval

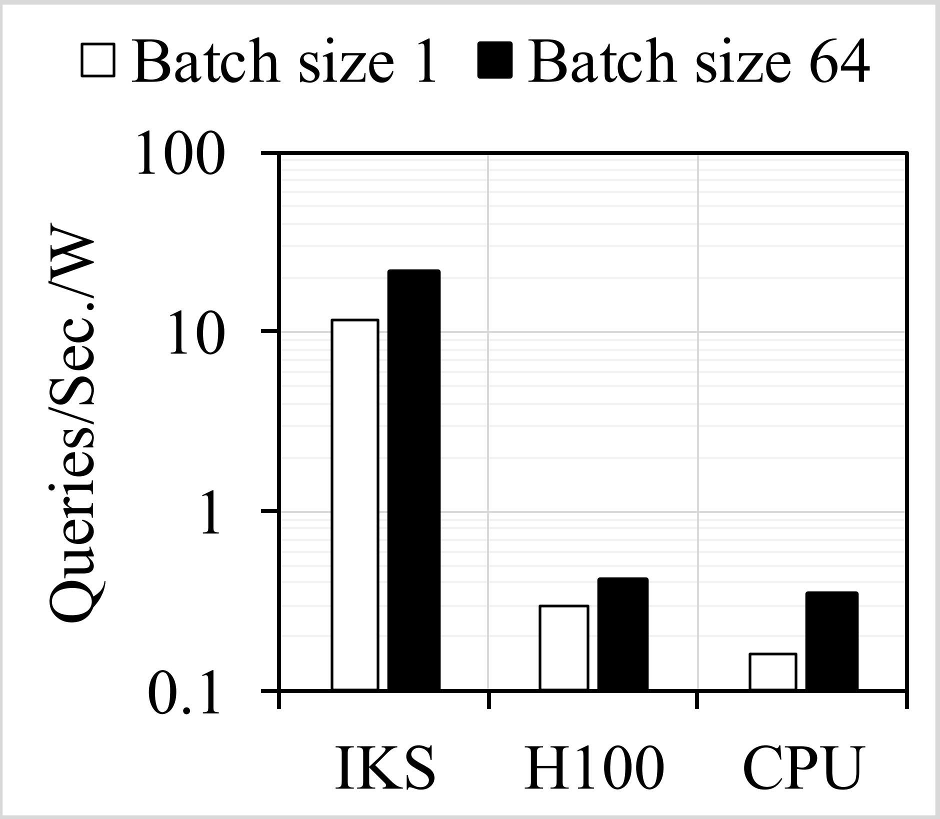
- IKS speedup:
  - Up to 13.5x over CPU
  - Up to 4.6x over 1x GPU
- 512 GB capacity enables large-scale retrieval

IKS vs. CPU and H100 GPU (Exhaustive Search)



Area and Power Efficiency

- Each NMA: 3.4 mm^2 (TSMC 16 nm)
- Total area: 220 mm^2
- LPDDR: 34.7 *W*
- Entire IKS power consumption
  - Batch size 1: 35.2 *W*
  - Batch size 64: 65.0 *W*



Throughput Retrieval Efficiency Comparison
(Higher is Better)

Contributions

- We showcased the system-level interactions of retrieval quality
- We leveraged algorithm-hardware co-design to build a high-quality retrieval accelerator, accelerating RAG
- We showcased the utility of CXL for the interface of high-capacity accelerators

For more information visit Alian Research Group at
https://arg.csl.cornell.edu