HyQA: Hybrid Near-Data Processing Platform for Embedding based Question Answering System

Shengwen Liang[1,2,3], Ziming Yuan[1,2], Ying Wang[4,2,3,5], Dawen Xu[7], Huawei Li[1,2,6], Xiaowei Li[1,2,3]
[1] SKLP, Institute of Computing Technology, CAS, Beijing. [2] University of Chinese Academy of Sciences, Beijing.
[3] Zhongguancun National Laboratory, Beijing. [4] CICS, Institute of Computing Technology, CAS, Beijing.
[5] Zhejiang Laboratory, Zhejiang, China. [6] Peng Cheng Laboratory, Shenzhen, China. [7] Seehi Microelectronic Corp. Ltd.
Email: {liangshengwen, yuanziming22s, wangying2009, lihuawei, lxw}@ict.ac.cn, xudawen@seehi.com.

***Abstract*—An Large Language Model (LLM)-based question-answering (QA) system has gained attention for its conversational ability. However, domain knowledge limitations, time lag, high training costs, and security concerns suggest building on-premise QA systems with embedding techniques. However, deploying embedding-based QA systems on existing GPUs or domain-specific accelerators is sub-optimal as they only address high computation costs and ignore large memory footprint and data movement costs, which impact response latency and user experience.**

To address these issues, we propose a hybrid near-data processing platform, HyQA, which collaboratively optimizes response latency, memory footprint, and data movement cost by exploiting the benefit of near-memory and near-storage computing simultaneously. First, HyQA analyzes computational patterns of sub-tasks in embedding-based QA systems, tailors domain-specific hardware accelerators, and assigns suitable computational paradigms. Second, these dedicated accelerators are designed to communicate directly with flash memory, avoiding additional data movement. The experiment shows that HyQA significantly improves performance and reduces energy over CPU, GPU, Cognitive SSD, and DeepStore platforms.

***Index Terms*—Near data processing, Embedding, Question answering, In-memory computing, In-storage computing**

I. Introduction

Question-answering (QA) [1] systems based on transformer-based LLMs (such as GPT) are known for their superior performance in question-answering. However, they could be more specific in topic breadth and timeliness of training data, resulting in negative and confabulated answers to domain-specific questions. Retraining and fine-tuning is a costly approach to learning knowledge due to the extremely high complexity of the model. Thereby, embedding-based QA system is becoming the mainstream solution for building on-premise QA systems.

The embedding-based QA system has three stages: (1) the **embedding building** stage generates an embedding vector database for each passage in the corpus. (2) the **search** stage recalls top-k relevant passages by evaluating the similarity between the question and the passages. (3) the **ask** stage integrates the top-k relevant passage with the input question and processes them for the final precise answers using transformer-based LLM. In this way, the quality of answers is improved, and the fine-tuning cost can be reduced as the up-to-date topics in the passage are covered in the QA system.

Information security and privacy concerns have driven the QA system to deploy on edge devices with real-time requirements. However, a transformer-based QA system is hard to apply to the edge device with GPUs or domain-specific accelerator due to: (1) ***high computation cost.*** Firstly, the QA system with transformer-based LLM suffers from high computational costs for inference. For example, the execution time of the GPT-2 model is 370ms on a TITAN Xp GPU and climbs to 43s on Raspberry Pi ARM CPU [2]. Secondly, indexing large-scale embedding vectors in the search stage requires immense computation and memory access overhead. For instance, querying a vector with 768 dimensions on a billion-scale dataset using the Intel CPU consumes about 50ms. (2) ***large memory cost*** comes from the parameters of LLM, large-scale corpus, vectors, and vector index. For instance, the GPT-3 with 175B parameters cannot be accommodated by NVIDIA H100 GPU, and the private knowledge corpus also requires a large storage capacity for vector index [1].

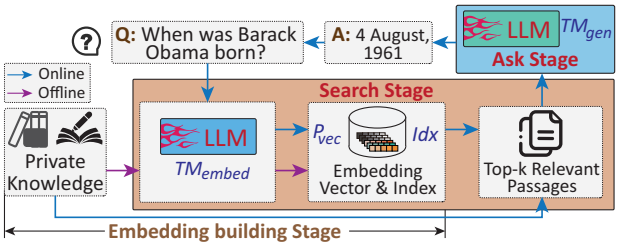

Fig. 1. The architecture of embedding based QA system. LLM: Large Language Model.

Previous works [2]–[6] improve the performance of transformer-based LLM QA systems by optimizing algorithms or customizing accelerators. However, they still struggle with large-scale corpus and introduce high data movement overhead because the limited memory capacity of edge devices introduces frequent data movement between low-level storage and main memory. Thereby, near-memory computing is a promising approach for reducing data movement, but it is limited by memory capacity due to the manufacturing process. In this case, near-storage computing is a better approach as it can move the computing unit into the storage device with sufficient space. However, only adopting near-storage computing is a non-trivial task due to insufficient bandwidth.

To address these issues, we profile the bottleneck of the transformer-based QA system and present the HyQA, a hybrid near-data processing platform for the QA system. The main contributions are:

1) HyQA combines DRAM-based near-memory computing with near-storage computing to provide sufficient bandwidth and reduce the data movement overhead.

2) HyQA uses a transformer and a vector search engine

979-8-3503-4859-0/DATE24/© 2024 EDAA

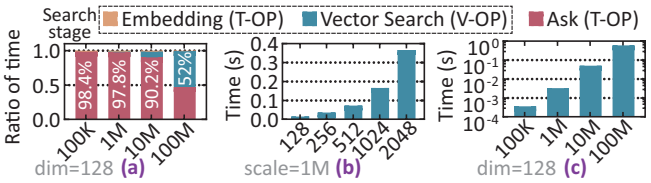

Fig. 2. (a) Execution time breakdown of QA system (EM-M+GPT) on GPU. The X-axis represents the scale of the vector database. (b) Execution time w.r.t vector dimension on CPU and (c) vector database scale on GPU.

to boost search and ask stages. These engines communicate directly with flash memory, avoiding additional data movement.

3) We implement a HyQA FPGA prototype and a simulator using SimpleSSD [7] to evaluate the response latency and energy efficiency. Experimental results show that HyQA achieves an average 16.7×, 8.59×, 19.35×, and 3.02× speedup, and 24.21×, 14.32×, 10.75×, and 2.34× energy efficiency over CPU, GPU, Cognitive SSD, and DeepStore, respectively.

II. Background

A. Preliminaries

Formally, the embedding-based QA system is defined (Fig. 1) as follows: Given a corpus that contains N documents, $\{d_0, ..., d_{n-1}\}$. Each d is split into multiple passages. Assuming corpus contains M passages $P = \{p_0, ..., p_{M-1}\}$. Each passages p_i can be viewed as a sequence of tokens $w_0^i, ..., w_{|p_i|-1}^i$. In the embedding building stage, all M passages are mapped to embedding vectors $P_{vec} \in \mathbb{R}^{M \times D}$ using a transformer-based embedding model TM_{embed}, where D is vector dimension. Then, P_{vec} are used to build an index Idx using an approximate nearest neighbor search (ANNS), also called vector search. In the search stage, given a question q, its embedding vector Q_{vec} is obtained using TM_{embed}. After that, the $topk$ similarity passages P_{topk} are extracted based on Idx using a vector search algorithm. Then, the input question q and the $topk$ passages P_{topk} are combined as the input sequence of transformer-based generative model TM_{gen} in the ask stage to get the answer.

B. Related Work

Software optimization. The bottleneck of the QA system comes from the computing-consuming transformer-based LLM and the memory-consuming embedding vector search. Thereby, recent efforts focused on (1) *Model optimization* aims to reduce the computational complexity of transformer-based LLM. For instance, a knowledge distillation method [6] is proposed to improve model inference efficiency. (2) *Index compression* removes redundancies in the embedding vector and its index. For example, mapping the embedding vectors to hash space can reduce the memory footprint [5]. However, the performance of these methods is still bounded by the underlying devices.

Hardware optimization. Customizing the accelerator is a promising solution to boost the performance of the embedding-based QA system. For instance, [2] presents an attention accelerator to boost transformer-based LLM such as BERT and GPT. [8] focuses on the acceleration of PQ-based vector search. However, these solutions only focus on a single component rather than the whole QA system. Simply integrating them fails to alleviate the latency and energy overhead incurred by the data movement overhead.

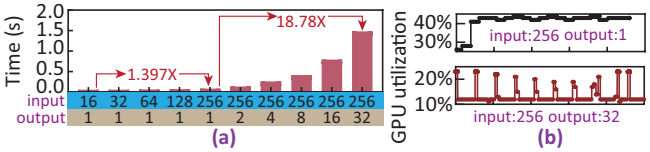

Fig. 3. (a) Execution time of GPT-large on GPU. (b) GPU utilization on GPT-large.

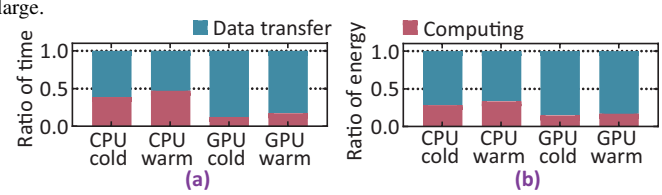

Fig. 4. (a) Time breakdown. (b) Energy breakdown. Cold: start the task from power off. Warm: start performance evaluation after a period of execution.

Near Data Processing contains two main paradigms: *near memory computing (NMC)* and *near storage computing (NSC)*. The former places the computing core close to the memory cell to exploit the memory bandwidth. For example, TransPIM is [3] tailored for transformer acceleration using NMC. However, NMC is limited by technology challenges in integrating and memory capacity. Thereby, NSC that places the computing core close to high-capacity storage devices is a promising solution. For instance, DeepStore [9] and Cognitive SSD [10] customize an in-storage dedicated accelerator. Unfortunately, they still fail to perform the QA system efficiently as the distinct operations within the transformer and vector search.

III. Motivation

A. Performance profiling

We profile the embedding-based QA system on a Xeon 4126 CPU with an A100 GPU. The Wikipedia is used as the knowledge corpus. The embedding model TM_{embed} and the generative model TM_{gem} are listed in Table III. NSG [11] algorithm is used to perform vector search as its superior performance.

The necessity of hardware acceleration. Fig. 2(a) shows that both transformer operator (**T-Op**) and vector search operator (**V-Op**) occupy a significant amount of execution time. Specifically, the ask stage dominates the execution time (98%) when the vector database is small. This is because (1) the ask stage needs to handle more tokens than the search stage. (2) the under-utilization of GPU when handling large output (see Fig. 3). With the increase of vector dimension and vector database size, the time of vector search (**V-Op**) accounts for over 52% (Fig. 2(b) and Fig. 2(c)). More seriously, the graph-based **V-Op** heavily relies on the graph structure and exhibits irregular memory accesses, resulting in low efficiency when deploying on CPU and GPU. In this case, designing the specific hardware accelerator is a promising solution.

B. Data movement cost

To profile the overhead incurred by data movement, we construct a CPU/GPU+DRAM+SSD system to profile the response time of the QA system under different scenarios, including cold start and warm start. The memory of the GPU is limited to 4GB to simulate scenarios where TM_{gem} cannot be placed on one GPU. Fig. 4(a) and Fig. 4(b) illustrate the response time

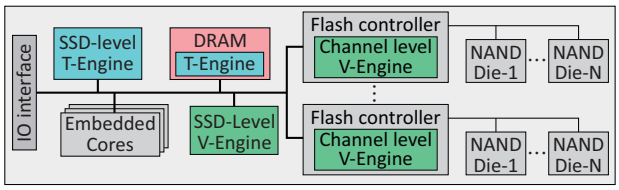

Fig. 5. The architecture of HyQA.

and energy breakdown of the QA system. It can be observed that around 60∼85% of total response time is spent on the data transfer, which indicates the QA system performance is predominantly limited by data movement.

The necessity of combining NMC to NSC. T-Op and **V-Op** are operators where computing-intensive and memory-intensive coexist. Specifically, computing-intensive **T-Op** involves complex data movement operations like transpose and concatenation. [2] figures out data movements account for 73% execution time. **V-Op** contains memory-intensive graph traversal phase and compute-intensive distance calculation phase. In this case, NMC is a promising solution. However, due to the limited memory capacity caused by chip technology, only adopting NMC still suffers from frequent data movement between storage devices, CPU memory, and device memory. Hence, combining NMC and NSC is a promising approach, as the former can provide sufficient bandwidth, and the latter, which possesses enough storage capacity, can shorten the data movement path.

IV. HyQA Design

A hybrid near-data processing platform, called HyQA (Fig. 5), is designed to enhance the performance of the transformer operator (T-Op) and vector search operator (V-Op), while also meeting the significant memory capacity demands and reducing the overhead of massive data movement. HyQA leverages the benefits of near-memory computing and near-storage computing, and consists of two major engines: the transformer engine (T-Engine) and the vector search engine (V-Engine), to boost the performance of T-Op and V-Op.

A. Transformer engine

The Transformer engine aims to execute T-Op in the search and ask stages efficiently. It is placed at SSD-level to exploit internal bandwidth and cache transformer parameters in DRAM. To address the limited capacity of internal DRAM, the on-chip memory of T-Engine interacts directly with multiple flash channels to bypass unnecessary data movement.

Architecture. Transformer engine (Fig. 6) includes two key modules: the far-memory module and the near-memory module. The far-memory module is implemented based on ASIC-based logic and mainly consists of the tensor and vector units. *Tensor unit* aims to perform matrix-vector and matrix-matrix multiplication (GEMM), which is the primary operation in transformer models. It contains multiple computing cores. Each computing core contains multiple tree-based multiplier-accumulators (MACs) to exploit the parallelism of GEMM. All tree-based MACs share a unified buffer. Each tree-based MAC has a local buffer (LB). *Vector unit* supports element-wise vector operations, including arithmetic and exponential

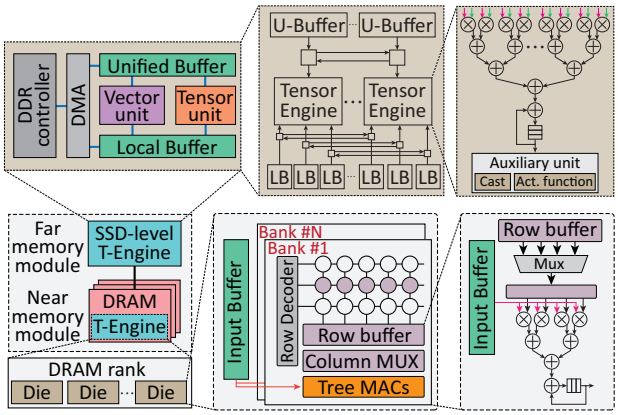

Fig. 6. The micro-architecture of Transformer Engine.

operation. The DMA engine acts as a bridge among the DRAM, on-chip buffer, and NAND flash chip and supports the transposed operator required by the attention operator.

The near-memory module aims to harvest the internal bandwidth across the banks within the DRAM. To this end, as shown in Fig. 6, each bank peripherals integrate a tree-based multiplier-accumulator implemented using DRAM technology to perform the GEMM and a private result buffer to hold the computed result. Considering the constraint of power and area overhead, the tree-based MACs are designed to receive the partial data from the row buffer as input instead of the width of the entire row buffer. To improve data reuse and minimize area overhead, the input buffer that is responsible for receiving the input vector sent from the far-memory module is shared by tree-based MACs across all banks. Given a matrix-vector multiplication, the value of the input vector is broadcast to all banks. The matrix is divided into sub-matrices that align with the number of banks. The row of sub-matrix is placed across the bank in an interleaved manner. In the calculation process, each row of sub-matrix and input vector data are calculated by tree-based MACs to generate the final result. After that, the far-memory module fetches the final result from the result buffer of each bank. In this way, we can offload the GeMM into the DRAM to reduce data movement overhead and boost the acceleration of the transformer operation.

Dataflow. Taking the transformer block as an example (Fig. 7). It includes three key layers: a fully-connected layer, a self-attention layer, and a feed-forward layer. The fully-connected layer receives an input sequence with L tokens to generate query Q, key K, and value V using the weight matrix W_Q, W_K, and W_V. These operations involve massive data movement. Thereby, we apply the head parallelism strategy to offload the fully-connected layer to the near-memory module to calculate the Q, K, and V in a head-wise manner. Considering the self-attention operation needs to perform data synchronization to gang the result of each head. Thereby, we transfer Q, K, and V to the far-memory module to perform the self-attention operation as it includes complex operations such as Softmax, which is unfeasible for the near-memory module. Similar to the fully-connected layer, due to the weight parameters of the feed-forward layer taking the majority (around 66%) of the transformer block, we opt to offload the feed-forward layer to

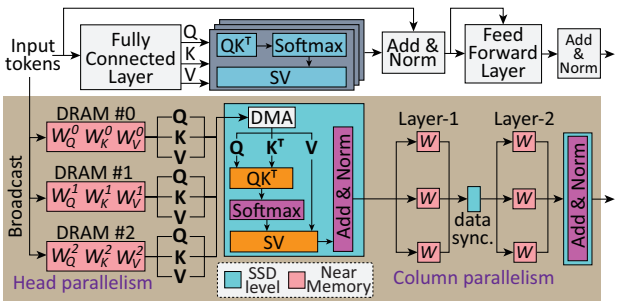

Fig. 7. The dataflow of transformer on T-Engine. DMA: Direct memory access. SV: multiplying the result of Softmax by value V.

the near-memory module. Specifically, the feed-forward layer's weight parameters are divided column-wise and then distributed to each DRAM chip for exploiting computing parallelism. Therefore, each DRAM chip's calculation result must be completed and synchronized before the next operation.

Algorithm 1 Distributed Graph-Based Vector Search

Input: Embedding vectors $\mathcal{X} \in \mathbb{R}^{N \times D}$, a query vector$q \in \mathbb{R}^{D}$, result size k, channel number C
Output: k nearest neighbors of a query q
1: Sub-vectors $\mathcal{X}_{sub}[1..C]$ = split($\mathcal{X}$) ▷ Offline
2: Each Sub-vector $\mathcal{X}_{sub}$ is organized to a KNN graph G ▷ Offline
3: **for** i in $1..C$ **do**
4: topk[i] = GraphSearch($\mathcal{X}_{sub}[i]$, $G[i]$, q, k) ▷ Online, Channel-level
5: **end for**
6: result = MergeTopk(**topk**, k) ▷ Online, SSD-level

B. Vector search engine

A graph-based vector search algorithm is employed in the vector search engine due to its excellent performance on large-scale vector databases. The HyQA design includes multiple channel-level V-Engines to exploit parallelism and an SSD-level V-Engine to consolidate results. The design is necessary because graph-based vector search relies heavily on graph structure, which results in dynamic and irregular data access patterns. Placing the V-Engine only at the SSD level could lead to under-utilization of bandwidth. As a result, the channel-level V-Engine is designed to use a divide-and-conquer approach to improve bandwidth utilization and reduce retrieval latency.

Algorithm. To exploit the internal channel-level parallelism of SSD, we adopt a distributed graph-based vector search algorithm in Alg. 1. Assume the embedding vector of the corpus is $\mathcal{X} \in \mathbb{R}^{N \times D}$ and SSD has C channels. The vectors $\mathcal{X}$ are uniformly divided into C parts. N/C vectors are assigned to a dedicated NAND flash channel and striped across flash chips. To boost the index time, each sub-vectors $\mathcal{X}_{sub} \in \mathbb{R}^{(N/C) \times D}$ is organized into a graph G, where each vertex corresponds to an instance in $\mathcal{X}_{sub}$ and each edge represents the similar association between the source vertex and the destination vertex. As shown in Alg. 1 and Alg. 2, the arrived query q is first distributed to each channel. Then, each channel performs a vector search in the graph G from the navigational vertex $p \in P$ to iteratively check neighbors' neighbors in the graph until the final top-K true neighbors of the query q are reached. Finally, the top-k results of each channel are merged to generate final top-K results.

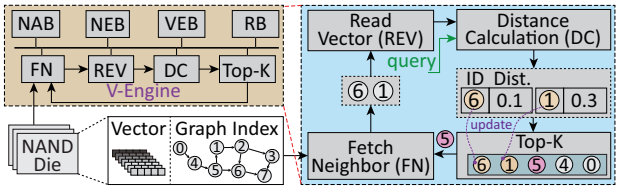

Fig. 8. The micro-architecture of a channel-level V-Engine.

Architecture. Alg. 2 depicts the design of a channel-level V-Engine that supports four key operations:

- *FN:* fetch the neighbor vertices according to the edge information of a vertex at the top of the priority queue.
- *REV:* read the vector according to fetched neighbors.
- *DC:* calculate the distances from vertices to the query.
- *Top-k:* sort the vertices in ascending order by the distance calculation result.

Fig. 8 shows the architecture of the channel-level V-Engine. The navigational vertex buffer (NAB) is used to store the entry vertices that are accessed at startup. The neighbor buffer (NEB) and vertex buffer (VEB) are designed to store the neighbor list in an adjacency list format and the required embedding vectors, respectively. Since distance computation dominates the execution time, the distance calculation module is designed to exploit dimension-level parallelism to boost the acceleration of distance calculation. The top-k module is implemented based on a hardware priority queue that facilitates a binary-heap-like structure for high throughput design. The top-k vertices are stored in the result buffer (RB).

Algorithm 2 Graph Search

Input: Embedding vectors $\mathcal{X}_{sub}$, Graph G with navigational vertex set P, a query vector $q \in \mathbb{R}^{D}$, result size k, Candidate pool S with size L
Output: k nearest neighbors of a query q
1: Candidate pool $S = \emptyset$ and $S.add(P)$
2: **while** $j < L$ **do**
3: j = the index of the first unchecked vertex in S
4: mark S_j as checked
5: **for** all neighbor $nbor$ of S_j in G **do** ▷ FN stage
6: Vector $E_{nei} = \mathcal{X}_{sub}$[nbor] ▷ REV stage
7: $dist = \phi(q, E_{nei})$; ▷ DC stage
8: $S.add(\{nbor, dist\})$
9: **end for**
10: Sort S in ascending order of the $dist$ value ▷ Top-k stage
11: $S.resize(L)$ when $S.size() > L$
12: **end while**
13: return the first k vertices in S

V. Evaluation

A. Experimental setup

Implementation. To evaluate the real-world platform, we implement a HyQA FPGA prototype with an OpenSSD [12] platform that only employs a near-storage computing paradigm, denoted as H-FPGA. To smash the bandwidth limitation of the FPGA prototype, we build a HyQA simulator only using SimpleSSD [7] that simulates near-storage computing, denoted as H-SIM-S. After that, we further integrate Ramulator-PIM [13] that models near-memory computing into H-SIM-S to evaluate the performance of HyQA, denoted as H-SIM-F. The near-memory module area estimation is done through

TABLE I
HARDWARE BASELINE.

	DFX [4]	Newton [15]	ZipNN [16]	VStore [17]
Data type	FP16	BF16	BF16	FP16
Computing unit	1088MACs	4096MACs	64MACs	512MACs
Frequency	200MHz	500MHz	200MHz	200MHz
Peak performance	435.2GOPS	4096GOPS	25.5GOPS	204.8GOPS
Peak Memory Bandwidth	HBM: 460GB/s DRAM: 38GB/s	HBM: 460GB/s	11GB/s	107.3GB/s
SSD Bandwidth	-	-	2.4GB/s	76.8GB/s

TABLE II
SYSTEM BASELINE.

	Cogn.SSD [10]	D.Store [9]	H-FPGA	H-SIM-S	H-SIM-F
Compute unit	256 MACs @100MHz	2048 MACs @800MHz	640 MACs @200MHz	3072 MACs @800MHz	7168 MACs @800MHz
On-chip memory	1.55 MB	8 MB	1.89 MB	4.28 MB	12.28 MB
Mem capacity	1 GB	32 GB	1 GB	32 GB	32 GB
Peak Mem BW.	8.5 GB/s	107.3 GB/s	8.5 GB/s	107.3 GB/s	460 GB/s
SSD BW.	1.4 GB/s	3.2 GB/s	1.4 GB/s	32 GB/s	32 GB/s
Internal BW.	1.4 GB/s	25.6 GB/s	1.4 GB/s	76.8 GB/s	76.8 GB/s

TABLE III
MODEL INFORMATION.

	Model Name	Vector Dim.	Parameters	Flops*
Embedding Model	qa-MiniLM-L6-cos-v1 [18] (EM-B)	384	33M	0.066B
	qa-distilbert-cos-v1 [18] (EM-M)	768	66M	0.1321B
	qa-mpnet-base-dot-v1 [18] (EM-L)	768	109M	0.2173B
Generative Model	gpt-base [18] (GPT)	-	124M	0.247B
	gpt-medium [18] (GPT-M)	-	355M	0.708B
	gpt-large [18] (GPT-L)	-	775M	1.547B
	gpt-xl [18] (GPT-XL)	-	1559M	3.114B

* represents flops per token.

CACTI-3DD [14] on a 22nm technology. The area and power consumption of the T-Engine and V-Engine of the far-memory module are estimated in Verilog under TSMC 65nm technology and then scaled to 22nm to match the memory technology. All solutions utilize a 16-bit float point.

Hardware Baseline. Table I shows the information of the hardware baseline used in our evaluation. We implement an ASIC-based DFX based on [4]. Newton [15] is used as a near-memory baseline. To evaluate the performance of V-engine, we select ZipNN [16] and VStore [17], both of which are designed as the in-storage vector search solution.

System Baseline. Table II shows the parameters of system baselines. It includes two baselines: 1) CPU platform is Xeon 4126@2.1GHz processor and 32GB DRAM and 2) GPU platform is equipped with NVIDIA A100 and 80GB HBM2. We also compare HyQA with Cognitive SSD [10] (Cogn.SSD in Table II) and DeepStore [9] (D.Store in Table II), the in-storage intelligent search solutions. We implement some necessary unsupported operators of a QA system in the embedded CPUs of Cogn.SSD and D.Store.

Benchmarks. Table III shows the typical transformer-based embedding models and generative models used in our evaluation. We adopt English Wikipedia [1] as the knowledge source, which contains 21 million passages after preprocessing. Each passage is represented by an embedding vector using the embedding model in Table III. We select a typical QA benchmark SQuAD (SD) [1] to evaluate the HyQA system in terms of response time and energy consumption. In addition, we select NSG [11] algorithm to perform vector search, which is the state-of-the-art graph-based algorithm.

B. Accelerator performance comparison

Power&Area. The T-Engine of H-FPGA has 512 MACs running at 200 MHz, and each channel-level V-Engine has

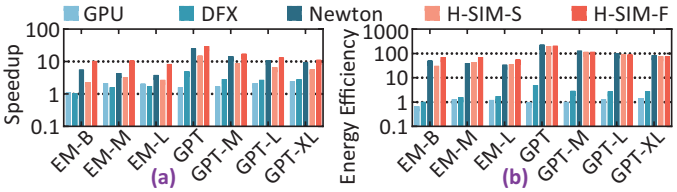

Fig. 9. Speedup (a) and Energy Efficiency (b) over the CPU baseline.

12 MACs at the same frequency. To balance the computing capability and bandwidth, the T-Engine and each channel-level V-Engine of H-SIM-S are adjusted to 2048 MACs and 12 MACs, respectively, incurring 9.51mm^2 area and 6.28W power overhead. The T-Engine in H-SIM-F combines near-memory computing and near-storage computing. The near-memory module contains 16 channels, and each channel has 16 banks. Each bank is equipped with 16 MACs. The total area and power of T-Engine and V-Engine in H-SIM-F are 25.45 mm^2 and 14.73W, respectively. The active power of H-FPGA and H-SIM-F are ∼20Watt and ∼45Watt, respectively.

T-Engine performance comparison. Fig. 9 shows T-Engine in H-SIM-S is 4.36× and 2.62× faster than DFX and GPU on the generative model, while slower than GPU on the embedding model. This is because the embedding model processes input tokens simultaneously, allowing GPU to release its performance. The limited computing capability and data movement overhead make H-SIM-S inferior to Newton. Thereby, the performance gain of H-SIM-F with near-memory computing climbs to 7.21× and 5.84× compared to GPU and DFX. H-SIM-F also achieves 1.53× performance improvement than Newton as it employs the far-memory module and near-memory module, which can avoid intermediate data being written back to the DRAM cell and is more suitable for reducing operators. In terms of energy efficiency, H-SIM-F outperforms GPU, DFX, and Newton by 81.27×, 39.53×, and 1.38×, respectively. This is because H-SIM-F performs a reduction operator on an ASIC-based module, which avoids the unnecessary data traffic.

V-Engine performance comparison. We compare the V-Engine of H-SIM-F with the CPU, ZipNN, and VStore baseline in terms of latency. The CPU baseline is used to run the Faiss framework. Fig. 10 (a) illustrates H-SIM-F provides 2.29×, 18.66×, and 1.22× speedup over CPU-16, ZipNN, and VStore on average under different vector database sizes. Fig. 10 (b) illustrates H-SIM-F offers 1.98×, 9.61×, and 1.26× speedup over CPU-16, ZipNN, and VStore on average under different vector dimensions. The performance improvements result from the exploration of channel-level parallelism, which is ignored by ZipNN and VStore. Meanwhile, Fig. 10 (c) and (d) depict H-SIM-F achieves much better energy efficiency compared to baseline. The improvement mainly comes from (1) channel-level parallelism reduces the complexity of graph-based vector search and obtains lower execution time; (2) near-storage computing shortens data movement path and avoids unnecessary data access.

C. End to end performance comparison

We evaluate the end-to-end latency of the HyQA against baselines using simulated user requests sourced from the SQuAD dataset. Fig. 11 illustrates that H-FPGA is lower than

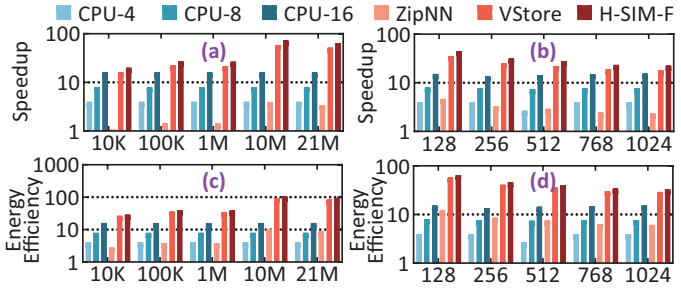

Fig. 10. Speedup and Energy Efficiency over the CPU-1 baseline, where the number is the number of threads. X-axis in (a) and (c) is the size of the vector database, and in (b) and (d) is the dimension of the vector. The dimension of the vector is 768 (a and c). The size of the vector database is 1M (b and d).

CPU and GPU because the transformer occupies more time on the resource-limited OpenSSD platform. H-FPGA shows a 2X speedup over Cognitive SSD as it customizes hardware architecture for the transformer and exploits the channel-level parallelism. Meanwhile, due to the transformer model involves massive memory-intensive operations, the limited hardware resource of the H-FPGA platform prevents us from allocating more computing resources. Therefore, Fig. 11 witness H-FPGA is inferior to DeepStore while H-SIM-S gains 4.84× and 1.51× speedup compared to H-FPGA and DeepStore. The reason is that H-SIM-S can provide sufficient computing resources and bandwidth over H-FPGA. Although H-SIM-S achieves superior performance improvement, it still needs to load data from off-chip memory at runtime. Frequent data movement constricts the performance of memory-intensive transformer and leads to high energy costs. Thereby, H-SIM-F with in-memory computing can avoid off-chip data movement overhead, achieving a 1.99× speedup compared to H-SIM-S.

Energy efficiency. The Intel RAPL and `nvidia-smi` are used to report CPU and GPU power, respectively. The power meter measures H-FPGA power. The simulator estimates the energy consumption of H-SIM-S and H-SIM-F. Fig. 12 shows H-FPGA achieves 3.12× improvement in energy efficiency over CPU thanks to the customized hardware architecture and the reduction of data movement between storage and CPU/GPU. Compared to Cognitive SSD, H-FPGA uses only 49% of the energy as it exploits channel-level parallelism and decreases redundant data transfer between NAND flash and internal DRAM device. H-SIM-S with high bandwidth achieves 3.45× energy efficiency over H-FPGA. The reason is that in addition to the execution time reduction, the accelerator within H-SIM-S shows higher energy efficiency than FPGA. As shown in Fig. 12, H-SIM-F is the most energy-efficient design and can achieve up to 16.76× better energy efficiency over CPU. This is because H-SIM-F integrated near-memory computing can further reduce data movement costs.

VI. Conclusion

The limited domain knowledge, the high fine-tuning cost, and information security and privacy promote individuality to construct an on-premise embedding-based QA system and deploy it on edge devices with real-time requirements. However, current solutions fail to meet the demands of QA systems on edge devices due to the overhead of massive data movement and

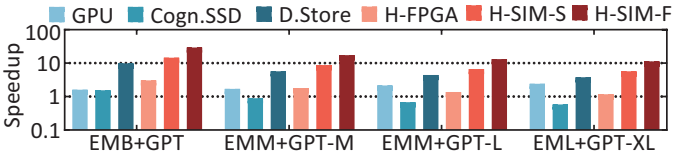

Fig. 11. Speedup over the CPU baseline.

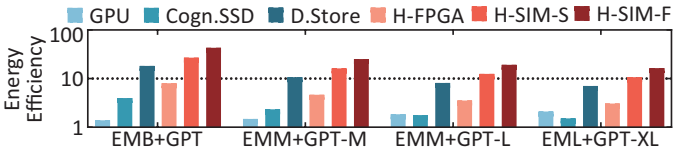

Fig. 12. Energy Efficiency over CPU baseline.

the expensive operations of the transformer. To address these issues, we propose HyQA, a hybrid near-data processing platform that combines the benefits of near-memory computing and near-storage computing to offload operations into SSDs. Our evaluation demonstrates that HyQA offers significant speedup and energy reduction compared to CPU, GPU, Cognitive SSD, and DeepStore solutions.

VII. Acknowledgment

This paper is supported in part by the National Natural Science Foundation of China (NSFC) under grant No.(62202453, 62090024, 61876173), and in part by the National Key Research and Development Program of China under grant 2018YFA0701502. This paper is also supported in part by China Postdoctoral Science Foundation 2022M713207. The corresponding authors are Shengwen Liang and Ying Wang.

References

[1] V. Karpukhin *et al.*, "Dense passage retrieval for open-domain question answering," in *EMNLP*, Online, Nov. 2020, pp. 6769–6781.
[2] H. Wang *et al.*, "Spatten: Efficient sparse attention architecture with cascade token and head pruning," in *HPCA*, 2021, pp. 97–110.
[3] M. Zhou *et al.*, "Transpim: A memory-based acceleration via software-hardware co-design for transformer," in *HPCA*, 2022, pp. 1071–1085.
[4] S. Hong *et al.*, "Dfx: A low-latency multi-fpga appliance for accelerating transformer-based text generation," in *MICRO*, 2022, pp. 616–630.
[5] I. Yamada *et al.*, "Efficient passage retrieval with hashing for open-domain question answering," in *ACL*, 2021, pp. 1–8.
[6] C. You *et al.*, "Knowledge distillation for improved accuracy in spoken question answering," in *ICASSP*, 2021, pp. 7793–7797.
[7] M. Jung *et al.*, "Simplessd: Modeling solid state drives for holistic system simulation," *IEEE CAL.*, vol. 17, no. 1, pp. 37–41, 2018.
[8] Y. Lee *et al.*, "Anna: Specialized architecture for approximate nearest neighbor search," in *HPCA*, 2022, pp. 169–183.
[9] V. S. Mailthody *et al.*, "Deepstore: In-storage acceleration for intelligent queries," in *MICRO*, New York, NY, USA, 2019, p. 224–238.
[10] S. Liang *et al.*, "Cognitive SSD: A deep learning engine for In-Storage data retrieval," in *USENIX ATC*, 2019, pp. 395–410.
[11] C. Fu *et al.*, "Fast approximate nearest neighbor search with the navigating spreading-out graph," p. 461–474, 2019.
[12] J. Kwak *et al.*, "Cosmos+ openssd: Rapid prototype for flash storage systems," *ACM Trans. Storage*, vol. 16, no. 3, jul 2020.
[13] G. Singh and other, "Napel: Near-memory computing application performance prediction via ensemble learning," in *DAC*, 2019, pp. 1–6.
[14] K. Chen *et al.*, "Cacti-3dd: Architecture-level modeling for 3d die-stacked dram main memory," in *DATE*, 2012, pp. 33–38.
[15] M. He *et al.*, "Newton: A dram-maker's accelerator-in-memory (aim) architecture for machine learning," in *MICRO*, 2020, pp. 372–385.
[16] G. Sun *et al.*, "Bandwidth efficient near-storage accelerator for high-dimensional similarity search," in *ICFPT*, 2020, pp. 129–138.
[17] S. Liang *et al.*, "Vstore: In-storage graph based vector search accelerator," in *DAC*, 2022, p. 997–1002.
[18] T. Wolf and other, "Huggingface's transformers: State-of-the-art natural language processing," 2020.