Reliable Hyperdimensional Reasoning on Unreliable Emerging Technologies

Hamza Errahmouni Barkam[1,*], Sanggeon Yun[1,*], Hanning Chen[1], Paul Gensler[2], Albi Mema[2], Andrew Ding[1]
George Michelogiannakis[3], Hussam Amrouch[2,4,5], and Mohsen Imani[1,†]
[1]University of California Irvine, [2]University of Stuttgart, [3]Lawrence Berkeley National Laboratory,
[4]Munich Institute of Robotics and Machine Intelligence, Technical University of Munich
[4]Chair of AI Processor Design, Technical University of Munich
[*]Equal contributions, [†]Correspondance: m.imani@uci.edu

2023 IEEE/ACM International Conference on Computer Aided Design (ICCAD) | 979-8-3503-2225-5/23/$31.00 ©2023 IEEE | DOI: 10.1109/ICCAD57390.2023.10323935

***Abstract*—While Graph Neural Networks (GNNs) have demonstrated remarkable achievements in knowledge graph reasoning, their computational efficiency on conventional computing platforms is impeded by the memory wall problem. To overcome these challenges, we introduce an innovative algorithm-hardware solution that harnesses the potential of hyperdimensional computing (HDC) for robust and memory-centric computation on computing in-memory (CiM) platforms. Departing from traditional graph neural networks, the proposed HDC reasoning model employs a symbolic approach to effectively encode graph entities and their relationships as high-dimensional neural activity. Complementing this approach is a customized Computing-in-Memory (CiM) architecture based on advanced Ferroelectric Field-Effect Transistor (FeFET) technology, which incorporates a precise characterization of non-idealities. This modeling enables the generation of an HDC-tailored model that faithfully represents the hardware architecture. Despite the non-idealities inherent in emerging CiM technologies, our platform demonstrates performance on par with traditional von Neumann architectures for substantial combinations of FeFET device parameters. Our solution overcomes FeFET CiM the increased non-idealities from down-scaled 3nm, operating effectively under all possible configurations when 50 graph edges are considered. Scenarios with less than 4-bit precision per FeFET device cannot handle graphs with more than 200 edges, whereas the 4-bit case can achieve a 90.3% graph reconstruction rate on the worst-case scenario of 80% of noise.**

I. Introduction

The remarkable achievements of artificial intelligence (AI) models can be attributed, in part, to their exceptional reasoning capabilities and the ability to derive novel interpretations from limited data[1, 2]. Reasoning as an essential capability of AI has been identified in [3, 4], which relies on prior knowledge and experiential learning. A Knowledge Graph (KG) is a graph-based data structure that leverages semantic relationships to organize and represent knowledge, facilitating efficient information retrieval and enabling complex reasoning tasks in a structured manner. Knowledge graph reasoning (KGR) uses these KGs and facilitates the derivation of previously undiscovered relationships among entities, thereby enriching the underlying knowledge graphs and empowering advanced applications. Notably, KGR models exhibit promising performance in various AI applications, including question-answering and recommendation systems[5, 6].

Graph Neural Networks (GNNs) have shown outstanding success in KG reasoning tasks[7]. These state-of-the-art algorithms utilize the graph structure to perform node and edge-level computations, enabling complex reasoning capabilities. However, the computational efficiency of GNNs on traditional computing platforms is hindered by the memory wall problem, resulting in a severe bottleneck of data transfers from the memory to the processing units[8].

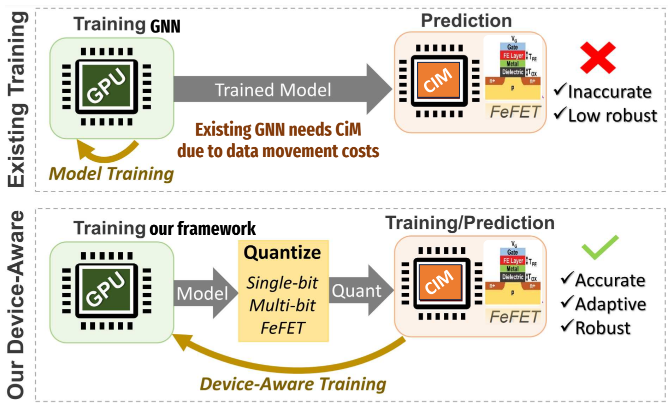

Fig. 1. Traditional computing on graph reasoning tasks is costly. Computing in memory is an emerging technology that brings the costs down exponentially but generates non-idealities that GNNs cannot handle. Our HDC-based Algorithm can perform with the anomalies introduced.

Brain-inspired computing represents a compelling paradigm that closely replicates the human brain's intricate workings[9–11]. Our brains exhibit remarkable characteristics such as efficient learning, resilience to noise, and works based on the in-memory computing principle[12–15]. In contrast, traditional von Neumann architectures, which separate memory and processing units, face limitations due to the inherent costs associated with data movement. To address these challenges, computing in memory (CiM) has emerged as a promising approach by seamlessly integrating memory and processing elements. CiM addresses the pressing concerns surrounding scalability and efficiency in conventional architectures by enabling parallel processing and minimizing data movement. Nonetheless, existing graph neural network (GNN) models do not inherently possess a memory-centric nature. They are vulnerable to the adverse effects of technology non-idealities that often present themselves as noise, leading to suboptimal quality and computational efficiency when deployed on emerging CiM architectures.

This paper introduces a novel algorithm technology co-design that revolutionizes graph reasoning algorithms by harnessing the potential of hyperdimensional computing (HDC) for robust and memory-centric computation on CiM platforms. Departing from conventional GNNs, our HDC reasoning model adopts a symbolic approach to effectively store graph entities and their relationships as high-dimensional neural activity. By optimizing the distribution of graph information across redundant high-dimensional vectors, our HDC reasoning algorithms facilitate highly parallel and fault-tolerant computation. This innovative approach, supported by the mathematical underpinnings of HDC, presents a promising solution for advancing knowledge graph reasoning and augmenting the capabilities of existing reasoning models.

Prior research has exhibited the intrinsic suitability of HDC models for optimization through hardware acceleration[16–20]. To enable efficient execution of HDC algorithms, we designed a tailored CiM architecture in order to optimize the HDC algebraic operations. The hardware platform presented in this study is based on advanced Ferroelectric Field-Effect Transistor (FeFET) technology and incorporates precise modeling of its non-idealities. This meticulous modeling allows for the generation of a specialized model that accurately represents the hardware platform. Leveraging the inherent advantages of CiM, such as having novel multi-bit CiM cells, parallel processing and reduced data movement, our algorithm effectively adapts to the inherent non-idealities prevalent in CiM devices such as scaling noise, temperature noise and low bit-precision.

Despite the challenges stemming from this emerging technology, our algorithm demonstrates comparable performance to traditional von Neumann architectures and improvements against GNNs. Our platform demonstrates superior performance in reasoning tasks compared to the Relational Graph Convolutional Network (RGCN) model, achieving an approximate 12.5% improvement in Mean Reciprocal Rank (MRR), which is a crucial metric for assessing graph reasoning. Furthermore, our framework exhibits 87-fold enhanced robustness against noise, highlighting its resilience in challenging environments. Notably, our solution overcomes significant limitations related to scalability and spatial constraints in FeFET CiM by successfully operating despite the non-idealities, including 3nm thick back-gate FeFET cells and elevated temperatures of 80 degrees Celsius, without any observable loss in performance. For graphs with up to 200 edges, the performance of 2-bit and 3-bits starts to fall down once we insert noises higher than 50%, but the 4-bit architecture is able to perform well once noise is introduced during learning phase, as the model learns to handle with a graph reconstruction rate of 90%.

II. Background & Related Work

A. Computing in Memory

Given the high number of matrix-vector multiplications in HDC algorithms, traditional von-Neumann processing architectures become in-adequate to satisfy the design space. CiM is a processing paradigm where the properties of the memory devices are exploited to perform calculations in-situ. CiM can be achieved by non-volatile memory (NVM) devices, such as FeFET or RRAM, configured in a crossbar array structure, which perform in-situ vector-matrix multiplications (VMM) in constant complexity, utilizing the analog domain.[21]. The operational procedure of the crossbar array is summarized as follows: (1) weight values are programmed as resistance levels into the NVM devices; (2) input values, encoded as voltage levels, are applied to the row bit lines; (3) the output of each NVM device, governed by Ohm's Law, is then summed in the column word lines by Kirchhoff's current law; (4) finally the summed current output must then be converted to a digital value as a registered output. Due to its in-situ properties, CiM-implemented VMM hardware is highly power efficient and suitable for low precision, area, and power applications [22]. However, this architecture is fundamentally limited by the necessary analog-digital conversions at the input/output interfaces between the crossbar unit to peripheral units or routing buses. Furthermore, computations performed in the analog domain by NVM devices, such as FeFet, must tolerate additional noise introduced by device variation, analog-digital conversion resolution, and limited device precision. For these reasons, HDC, through its inherent algorithmic redundancy and robustness, is prime candidate for acceleration with CiM using emerging technologies such as FeFet[23–25].

B. Graph Learning

The acceleration of graph-related algorithms, such as Graph Neural Networks (GNNs), using domain-specific accelerators (DSAs) such as Field-Programmable Gate Arrays (FPGAs) [26, 27], Application-Specific Integrated Circuits (ASICs) [28], and CiM [29], has garnered significant attention in recent times. Unlike traditional machine learning tasks, graph-based learning tasks typically involve large-scale datasets, necessitating the design of hardware accelerators. However, most previous studies have focused on accelerating graph neural networks and mining. More recently, the graph learning framework based on hyperdimensional computing has gained substantial interest [30–32].

In particular, a FeFET-based CiM hyperdimensional graph classification accelerator was demonstrated in the work by [31]. This study centered on accelerating HDC-based graph learning and compared it with previous graph neural network acceleration methods. However, the work did not mention any contributions towards brain-inspired graph reasoning acceleration [31]. On the other hand, [32] introduced the first hyperdimensional computing-based graph reasoning framework. The authors proposed a hyperdimensional graph reasoning model and a digital CiM hardware platform. Nevertheless, this work only explored basic digital CiM acceleration and paid limited attention to model quantization, error analysis, and analog CiM acceleration. Leveraging model quantization and ensuring robustness become imperative to achieve efficient acceleration of HDC-based graph models using high-speed analog CiM.

C. FeFET Basics

A FeFET is a type of transistor that uses a HfO_2-based ferroelectric layer (FE layer) in the gate stack to control the channel conductivity [33]. The FE layer is highlighted in red in Fig. 2 and comprises many domains that can be polarized individually [34]. Depending on their polarization, the channel conductivity changes, which is mapped to logic states. Additionally, the domains retain their polarization without a voltage making FeFET technology a multi-level non-volatile memory. To polarize the domains in the same direction, a write pulse of $-4\,V$ or $+4\,V$ is applied, typically with a duration of $1\,\mu s$ [33]. If the pulse has a lower voltage, not all domains are flipped, and the FeFET's conductance has an intermediate level. Consequently, a multi-level cell (MLC) is created. A small voltage is applied during a read operation, and the conducted current is measured. This voltage causes read disturbance since some domains could be flipped during a read [35].

Another challenge is the thickness of the FE layer, which is typically about $10\,nm$ [36]. A thicker layer increases the memory window, the difference between the low and highly polarized state, and a metric to maximize. On the one hand, a thicker layer also hurts the ferroelectric properties and prevents scaling the underlying transistor. On the other hand, a thinner layer reduces the number of domains, and each domain becomes more impactful on the overall polarization of the device. Because the domain's polarization change is a stochastic process,

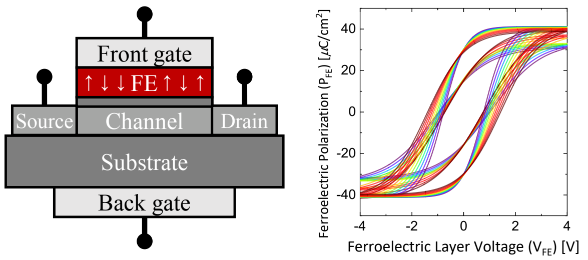
Front gate
↑↓↓FE↑↓↑
Source
Channel
Drain
Substrate
Back gate
Ferroelectric Polarization (P_FE) [μC/cm²]
Ferroelectric Layer Voltage (V_FE) [V]
40
20
0
-20
-40
-4
-2
0
2
4

Fig. 2. Adding a ferroelectric (FE) layer to the regular gate stack allows the channel conductivity to be controlled within a range known as the memory window. The size of this window can be increased by incorporating a back gate. The right image shows the hysteresis behavior of ferroelectric cells, which is influenced by bit precision, where higher precision amplifies hysteresis effects and lower precision diminishes them. The choice of precision depends on specific application requirements.

the variability increases for highly-scaled FeFET devices with fewer domains [37]. This results in a device that suffers more from non-idealities.

Advanced dual-gate FeFET: To address both challenges, a FeFET with an additional back gate has been recently proposed [38]. By introducing this back gate and reading through it, the domains in the FE layer are no longer disturbed. The FeFET is still written through the front gate to change the FE layer's polarization. In addition, the memory window is amplified by the body effect factor [39] if the FeFET is read through the back gate. A negative aspect is an increasing variability because of the increased distance between the channel and the back gate compared to the front gate [37].

III. Graph Encoding in Hyperdimensional encoding

This section describes the utilization of a hyperdimensional algorithm inspired by the workings of the human brain for the purpose of learning and memorizing the given graph. Additionally, adaptations to the algorithm were made to optimize its performance and accommodate the inherent non-idealities associated with FeFET technology, as depicted in Figure 3.

The initial phase of graph memorization involves initializing five key parameters, namely B, D, G, and M_c. B denotes the number of bits employed in the encoding process, while D represents the dimensionality of the hyperdimensional space utilized. The graph $G = (V, E)$ comprises a set of vertices V and a set of edges E that need to be memorized. Furthermore, the current matrix M_c encompasses the mapping values that facilitate the computation of similarities between graph elements. Subsequently, the encoding procedure is executed to encode the given graph into the hyperdimensional representation.

In the subsequent step, the model proceeds to encode the given graph G into a hyperdimensional space. To accomplish this, D-dimensional vectors, denoted as $\vec{H}_v \in \mathbb{R}^D$, are generated for each vertex $v \in V$. These vectors are sampled from a normal distribution, specifically from the set $\{\mathcal{N}(0,1)\}^D$. Subsequently, memory node hypervectors, represented as $\vec{M}_v \in \mathbb{R}^D$, are computed by aggregating the hypervectors of neighboring vertices for each vertex $v \in V$, as visually depicted in Figure4. In other words, for each edge $e_i = (v_i, u_i) \in E$, the hypervectors $\vec{H}_{v_i}$ and $\vec{H}_{u_i}$ of vertices v_i and u_i, respectively, are merged to form Mv and Mu. Finally, as illustrated in Figure 5, the graph memory $\vec{\mathcal{G}}$ is computed as the summation over all vertices $i \in V$ of the element-wise multiplication (denoted by

Algorithm 1 Graph Encoding Refinement

```
Input:
    $G(V,E)$: Undirected graph consisted of vertex set $V$ and edge set $E$ to encode
    $\vec{\mathcal{G}}$: Previously encoded graph memory
    $\vec{H}_v$: Vertex hypervectors
Output:
    $\vec{\mathcal{G}}'$: Refined graph memory
    $T$: Threshold
1: procedure GraphEncodingRefinement($G$, $\vec{\mathcal{G}}$, $\vec{H}_v$)
2:     $\vec{M}_v^{(1)} \leftarrow \vec{\mathcal{G}} * \vec{H}_v$
3:     for $i \in \{1,2,...,n-1\}$ do
4:         $\vec{M}_v^{(i+1)} \leftarrow \vec{\mathcal{G}} * \vec{H}_v - \eta \sum_{u \neq v} \vec{H}_v * \vec{H}_u * \vec{M}_u^{(i)}$
5:     end for
6:     $[\vec{H}_v^q,\ \vec{M}_v^q] \leftarrow [q(\vec{H}_v),\ q(\vec{M}_v^{(n)})]$
7:     $S_{pos} \leftarrow \{(v,u) \in V \times V | (v,u) \in E\}$
8:     $S_{neg} \leftarrow \{(v,u) \in V \times V | (v,u) \notin E\}$
9:     $T \leftarrow \frac{1}{2}\left(\frac{\sum_{(v,u)\in S_{pos}} \delta(\vec{M}_v^q, \vec{H}_u^q)}{|S_{pos}|} + \frac{\sum_{(v,u)\in S_{neg}} \delta(\vec{M}_v^q, \vec{H}_u^q)}{|S_{neg}|}\right)$
10:    for $i \in V$ do
11:        for $j \in V$ do
12:            if $\frac{\delta(\vec{M}_i^q, \vec{H}_j^q)}{D} < T$ and $(i,j) \in E$ then
13:                $\vec{M}_i \leftarrow \vec{M}_i + \alpha \vec{H}_j$
14:            else if $\frac{\delta(\vec{M}_i^q, \vec{H}_j^q)}{D} > T$ and $(i,j) \notin E$ then
15:                $\vec{M}_i \leftarrow \vec{M}_i - \alpha \vec{H}_j$
16:            end if
17:        end for
18:    end for
19:    $\vec{\mathcal{G}}' \leftarrow \sum_{i \in V} \vec{H}_i * \vec{M}_i$
20: end procedure
```

Algorithm 2 Graph Decoding

```
Input:
    $\vec{\mathcal{G}}$: Encoded graph memory
    $\vec{H}_v$: Vertex hypervectors
    $T$: Threshold
Output:
    $E'$: Edge set of reconstructed graph
1: procedure GraphDecoding($\vec{\mathcal{G}}$, $\vec{H}_v$, $T$)
2:     $\vec{M}_v^{(1)} \leftarrow \vec{\mathcal{G}} * \vec{H}_v$
3:     for $i \in \{1,2,...,n-1\}$ do
4:         $\vec{M}_v^{(i+1)} \leftarrow \vec{\mathcal{G}} * \vec{H}_v - \eta \sum_{u \neq v} \vec{H}_v * \vec{H}_u * \vec{M}_u^{(i)}$
5:     end for
6:     $[\vec{H}_v^q,\ \vec{M}_v^q] \leftarrow [q(\vec{H}_v),\ q(\vec{M}_v^{(n)})]$
7:     $E' \leftarrow \emptyset$
8:     for $i \in V$ do
9:         for $j \in V$ do
10:            if $\frac{\delta(\vec{M}_i^q, \vec{H}_j^q)}{D} \geq T$ then
11:                $E' \leftarrow E' \cup \{(i,j)\}$
12:            end if
13:        end for
14:    end for
15: end procedure
```

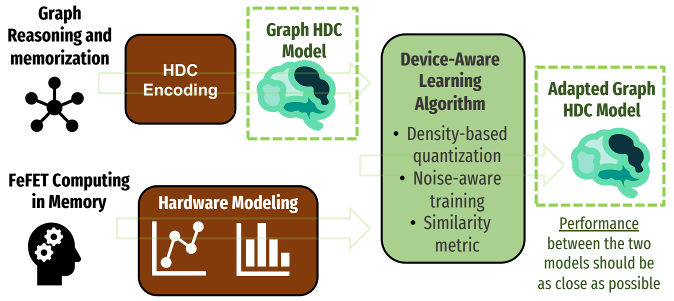
Graph Reasoning and memorization
HDC Encoding
Graph HDC Model
Device-Aware Learning Algorithm
• Density-based quantization
• Noise-aware training
• Similarity metric
Adapted Graph HDC Model
Performance between the two models should be as close as possible
FeFET Computing in Memory
Hardware Modeling

Fig. 3. General overview of our proposed framework. We begin by generating a graph hyperdimensional model on ideal conditions and modeling the non-idealities of the hardware. Then we proceed to combine both in order the exploit HDCs strengths to generate a compatible model with FeFET Computing in Memory architecture.

$*$) between the hypervector $\vec{H}_i$ and the corresponding memory node hypervector $\vec{M}_i$, resulting in $\vec{\mathcal{G}} = \sum_{i\in V} \vec{H}_i * \vec{M}_i$.

A. Node memory reconstruction

In order to retrieve node memory $\vec{M}_v$ using encoded graph memory $\vec{\mathcal{G}}$ and vertex hypervectors $\vec{H}_v$, iterative node memory reconstruction is conducted. The initial estimation $\vec{M}_v^{(1)}$ for node memory $\vec{M}_v$ is computed by $\vec{M}_v^{(1)} = \vec{\mathcal{G}} * \vec{H}_v$. However, the first estimation contains noise which is introduced by:

$$\begin{aligned}\vec{\mathcal{G}} * \vec{H}_v &= \vec{H}_v * \vec{H}_v * \vec{M}_v + \sum_{u\neq v} \vec{H}_v * \vec{H}_u * \vec{M}_u \\ &= \vec{H}_v * \vec{H}_v * \vec{M}_v + noise \approx \vec{M}_v.\end{aligned}$$

To cancel noise iteratively, the next estimation $M_v^{(n+1)}$ of $M_v^{(n)}$ is computed by $\vec{M}_v^{(n+1)} = \vec{\mathcal{G}} * \vec{H}_v - \eta \sum_{u\neq v} \vec{H}_v * \vec{H}_u * \vec{M}_u^{(n)}$ where $\eta \in \mathbb{R}$ indicates noise cancellation rate. Node memory reconstruction is conducted at the beginning of each encoding refinement iteration (Algorithm 1 line 4) and graph decoding process (Algorithm 2 line 4).

B. Graph memory refinement

Memory refinement is done in an iterative process to have a better encoding of $\vec{\mathcal{G}}$. A single iteration for this graph memory refinement process is described in Algorithm 1. For each memory refinement step, we update memory node hypervectors using every pair of vertices $(v, u) \in V \times V$ as follows:

$$\begin{cases} \vec{M}_v = \vec{M}_v + \alpha\vec{H}_u & \text{if } \delta(q(\vec{M}_v^{(n)}), q(\vec{H}_u))/D < T \text{ and } (v, u) \in E \\ \vec{M}_v = \vec{M}_v - \alpha\vec{H}_u & \text{if } \delta(q(\vec{M}_v^{(n)}), q(\vec{H}_u))/D > T \text{ and } (v, u) \notin E \end{cases}$$

where $\alpha \in \mathbb{R}$ indicates the refinement rate. M^c is used in δ, a function for computing similarity between the given hypervectors. At the end of each iterative refinement, $\vec{\mathcal{G}}$ is computed from updated $\vec{M}_v$.

C. Decoding of the graph

To decode the graph, we test for every pair of vertices $(v, u) \in V \times V$ to determine whether or not an edge (v, u) exists. The existence of an edge (v, u) is determined by $\delta(q(\vec{M}_v^{(n)}), q(\vec{H}_u))/D \geq T$ after getting estimated node memory $\vec{M}_v^{(n)}$ from $\vec{\mathcal{G}}$. If the inequality is satisfied, it considers the edge as a part of the graph. This graph decoding process is described in Algorithm 2.

D. Adaptation of graph memorization to Computing in Memory

Next, we present the adaptations undertaken to our pre-established algorithm in order to effectively harmonize it with the prevalent non-idealities inherent to the device.

1) Projection of the model: During the graph projection process, we proceed to ensure that the full-precision hypervectors $\vec{H}_v$ and $\vec{M}_v$ are transformed into B-bit precision representations to be compatible with the CiM architecture. This quantization operation is carried out within the model projection step, where the components of the hypervectors are quantized to B-bit symbols. Given that the feature values do not typically conform to a uniform distribution, we do not perform uniform quantization. Instead, the values of the HDC components are calculated, and the cumulative normal distribution function (cdf) is utilized for feature value quantization. As a result, the quantized vertex hypervectors $q(\vec{H}_v)$ and quantized memory node hypervectors $q(\vec{M}_v)$ are obtained from the original $\vec{H}_v$ and $\vec{M}_v$, respectively, as shown in Figure 6.

2) Similarity function setup: The HDC algebra relies on a similarity metric for its learning tasks. In our CiM architecture, the traditional dot product or cosine similarity operations are replaced by function that compares bit values within the device. This enables efficient similarity evaluations while considering the limited computational resources. Given two B-bit(s) hypervectors H_1 and H_2, current similarity $\delta(\vec{H}_1, \vec{H}_2)$ is computed using $\vec{d} = |\vec{H}_1 - \vec{H}_2|$ and M^c indicating current matrix containing mapping values for computing similarities. Now, the similarity is computed as follows.

$$\delta(\vec{H}_1, \vec{H}_2) = \sum_{i=0}^{D-1} M^c_{\vec{d}_i}$$

Note that $0 \leq \vec{d}_i < |M^c| = 2^B$.

3) FeFET noise application to the graph framework: For the purpose of applying noise, given a modeled probability distribution, the noise makes a bit-symbol v increase or decrease to $v+1$ or $v-1$. Each value in the quantized hypervector has the same probability of random value changing. For instance, if the random changing probability is p, a value v in the hypervector will be changed to the value of $v+1$ in $p/2$ probability and $v-1$ in $p/2$ probability. But, if $v-1$ is 0 or $v+1$ is $2^B - 1$, only an increase or decrease will be applied in p probability, respectively. The application of random value-changing noise can be described as follows:
The noise is introduced between the model projection and the inference steps, affecting the test set. The changed values persist until the subsequent model projection occurs. We study two different scenarios where the noise is introduced. The first one includes the noise during decoding, meaning the encoding process of the graph is not affected by value changes. Instead, the noise is applied to a copied model during the iterative refinement of graph encoding. The second one introduces the noise during the generation of hypervector $\vec{\mathcal{G}}$

E. Capacity of hardware-based HDC

Capacity plays a vital role in Hyperdimensional Computing (HDC), influencing its effectiveness and efficiency. It refers to the system's capability to process and manage a substantial volume of information within its high-dimensional vectors. A greater capacity enables HDC to represent and manipulate a larger number of distinct entities. The concept of capacity is crucial for accurately encoding, storing, and reasoning with complex data, including graphs and knowledge bases. In this section, we introduce a theoretical formulation and modified approach tailored to model quantization. In line with the approach described in [40], the memory capacity is defined as the information content of the memory, specifically the mutual information between the true nodes and the nodes that can be retrieved from the model $\vec{S}$. It is important to note that since the model is solely used for

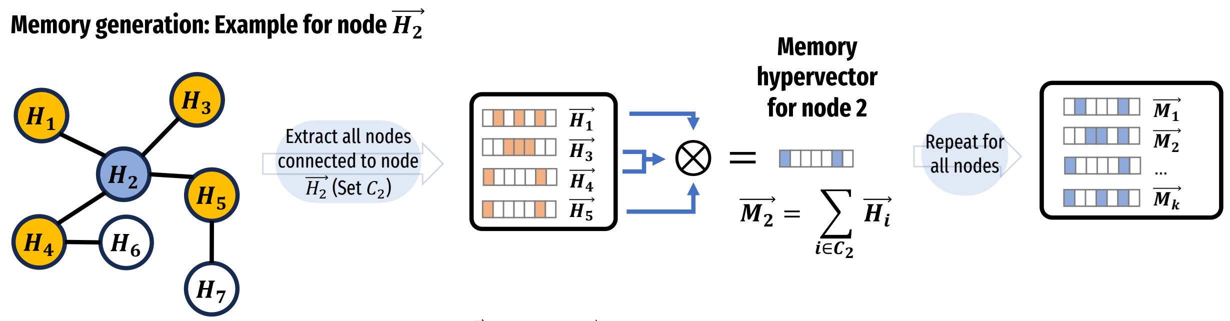
Memory generation: Example for node $\vec{H_2}$
Extract all nodes connected to node $\vec{H_2}$ (Set C_2)
Memory hypervector for node 2
$\vec{M_2} = \sum_{i \in C_2} \vec{H_i}$
Repeat for all nodes

Fig. 4. Example of memory generation for a specific Graph $\vec{G}$ and node $\vec{H_2}$. We achieve so by bundling all the graph hypervectors that are connected to the node.

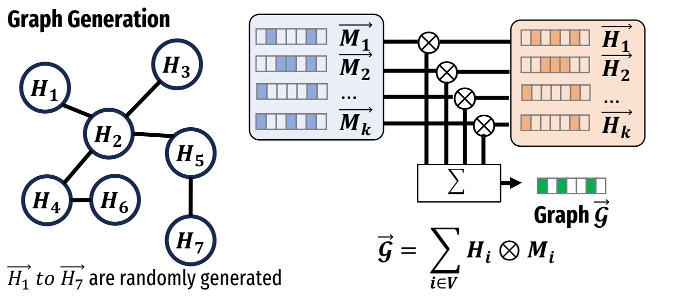
Graph Generation
$\vec{H_1}$ to $\vec{H_7}$ are randomly generated
Graph $\vec{\mathcal{G}}$
$\vec{\mathcal{G}} = \sum_{i \in V} H_i \otimes M_i$

Fig. 5. Once we generate all the memory node hypervectors for every node, we memorize the model by binding each node hypervector $\vec{H_i}$ with its respective memory vector $\vec{M_i}$.

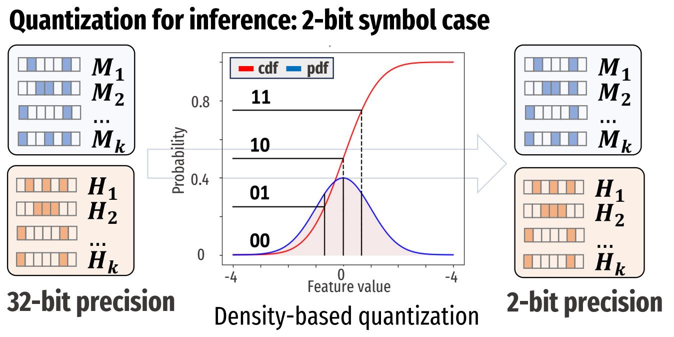
Quantization for inference: 2-bit symbol case
cdf
pdf
Probability
Feature value
32-bit precision
Density-based quantization
2-bit precision

Fig. 6. Projection step, achieved by gathering the values inside the dimensions of the hypervectors $\vec{H_i}$ and $\vec{M_i}$ and doing a density-based quantization given a specific bit-precision.

detection purposes, the analysis of mutual information focuses on the distribution of node membership rather than the nodes themselves. Let $\hat{s}$ be the random variable of the detector output and s be the membership of a query. For simplicity, let the support of $\hat{s}, s$ be $\{0, 1\}$, indicating undetected/detected for $\hat{s}$ and not present/present for s, respectively. Therefore, under fixed parameter p_s and threshold θ', the mutual information between the set of nodes $\{\vec{S}^{(i)}\}$ and the model $\vec{S}$ is

$$I(\{\vec{S}^{(i)}\}, \vec{S}) = D_{KL}(\Pr(\hat{s}, s) || \Pr(\hat{s}) \Pr(s))$$
$$= \sum_{i,j \in \{0,1\}} \Pr(\hat{s} = i, s = j) \log_2 \frac{\Pr(\hat{s} = i, s = j)}{\Pr(\hat{s} = i) \Pr(s = j)}$$

Where D_{KL} is the KL divergence [41]. By definition, $\Pr(s)$ is described succinctly by p_s ($\Pr(s = 1) = p_s$). $\Pr(\hat{s} = 1) = tp \cdot p_s + fp \cdot (1 - p_s)$ is the marginal probability with which the detector outputs "detected". The joint probability $\Pr(\hat{s}, s)$ can be computed by the conditional probability, whose values are the true/false positive/negative rates of the detector. The memory capacity is simplified as

$$I(\{\vec{S}^{(i)}\}, \vec{S}) = p_s(tp \log tp + (1 - tp) \log(1 - tp) - \log Z)$$
$$+ (1 - p_s)(fp \log fp + (1 - fp) \log(1 - fp) - \log(1 - Z))$$

where $Z = \Pr(\hat{s} = 1) = tp \cdot p_s + fp \cdot (1 - p_s)$, and θ' is implicit.

We analyze the impact of quantization using high-resolution quantization theory. As this quantization method takes into account the statistics of all hypervector components (D of them), and each component is quantized to the same amount of bits (encoding rate is fixed), it is classified as a fixed-rate D–dimensional quantizer. As a result, the quality of the quantizer $\delta_D(R)$ has an upper bound of

$$\delta_D(R) \cong M_D \beta_D \sigma^2 2^{-2R}$$

Where $\delta_D(R)$ is the operational rate-distortion function of the quantizer, and MSE measures the distortion. R is the bit rate, $Z_D(R)$ is the Zador-Gersho function, $M_D \xrightarrow{D \to 0} (2\pi e)^{-1}$ is Gersho's constant [42] that accounts for the least normalized moment of inertia of D–dimensional tessellating polytopes, $\beta_D \xrightarrow{D \to 0} (2\pi e)$ is Zador's factor [43], and σ is the standard deviation of the source (assumed gaussian). As suggested in [44], the high dimensionality of the quantizer allows close approximation of the constants. Furthermore, the performance of the D-dimentional quantization method converges to that of the optimal D-dimentional quantizer as D approaches infinity.

IV. Hardware Framework

In contrast to conventional hardware computing platforms like CPU and GPU, HDC models typically exhibit superior execution performance and energy efficiency when applied to computing in-memory (CiM) [45, 46]. Compared with digital CiM, crossbar array-based analog CiM shows more advantages on speedup and energy efficiency [47]. To enhance the speed of our design even further, we leverage an analog computer in-memory platform by mapping our design onto it. For the vertex memory hypervector reconstruction process, we write the equation as follows:

$$\vec{M}_i^{(k+1)} = \vec{H}_i \circ \vec{\mathcal{G}} - \vec{H}_i \circ \sum_{j \in V \setminus \{i\}} \vec{H}_j \circ \vec{M}_j^{(k)} \approx \vec{H}_i \circ \vec{\mathcal{G}} - \vec{H}_i \circ \sum_{j \in V} \vec{H}_j \circ \vec{M}_j^{(k)} + \vec{M}_i^{(k)} \quad (1)$$

To accelerate vector-to-vector element-wise multiplication and accumulation operations, we map equation 1 into crossbar array [48]. The analog CiM's architecture and schematic design are shown in Figure 7. (a). Here we cut the hypervector into **N** chunks with each chunk's dimension as **T**. Each processing unit (PU) has one crossbar array for the element-wise product (EP), as is shown in Figure 7. (b)

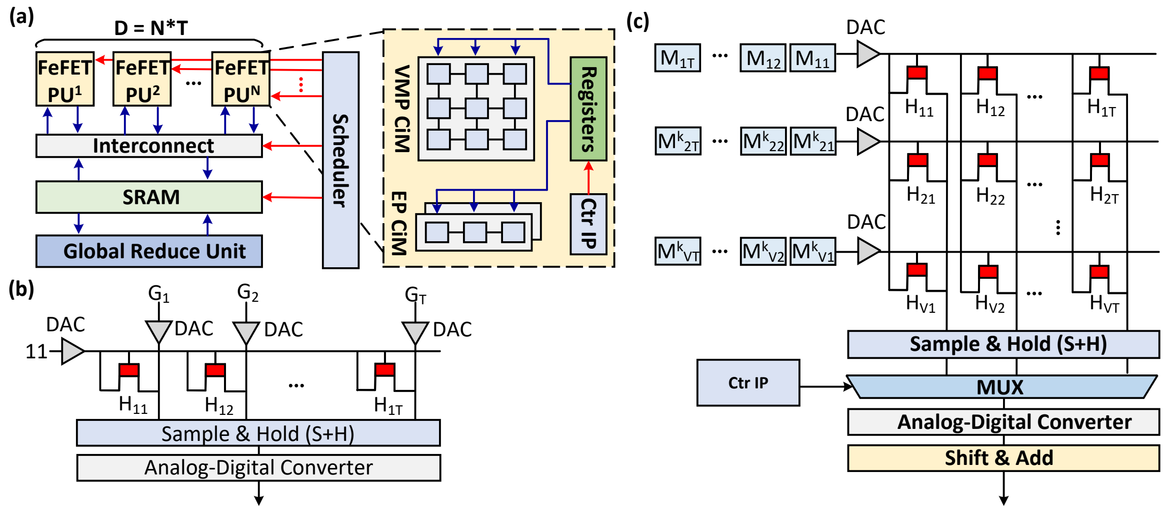

Fig. 7. (a) FeFET-based analog CiM accelerator targetting vertex memory hypervector reconstruction. (b) Vector to vector element-wise product (EP) CiM. (c) Vector-matrix product (VMP) CiM.

and one array for vector-matrix multiplication (VMP) 7. (c). After generating each vertex's memory hypervector, we use FeFET-based TCAM to accelerate the hamming distance computation between the query vertex hypervector and vertex memory hypervector when determining whether two vertices are connected. TCAM-based PIM is widely used by previous HDC accelerator works [20, 49].

The vector storage is divided into smaller FeFET-based crossbars. The values are applied to the horizontal word lines as the gate voltages of the FeFETs. Based on the FeFET's state, its conductance value is different, representing the multiplication. All FeFETs of a single column share a common wire connected to their drain terminal, combining the conductance values and performing the addition. To reduce unnecessary power consumption, the drain terminals of the unselected columns are tied to the ground. Only the active column is selected through a MUX and connected to the ADC. Since the crossbar is operated at 50 mV and the ADC at 800 mV, a current mirror [50] is added.

The ADC is based on the ladder principle using transistors, similar to the architecture described in [51]. The ADC includes buffers to rectify the output flanks and four full adders to convert the thermometer output representation into a binary representation.

V. Evaluation

A. Experimental Setup

The comprehensive framework encompassing graph learning and reasoning is successfully implemented utilizing the PyTorch deep learning library. To evaluate the performance of our framework, a rigorous and systematic assessment is conducted, employing well-established benchmark models such as the Relational Graph Convolutional Network (RGCN) as a baseline and FB15K-237 [52] as the dataset. We employ standard metrics commonly used in KGR tasks, as shown in Table I. These metrics include Mean Reciprocal Rank (MRR), Hits@1 (H@1), Hits@3 (H@3), and Hits@10 (H@10). MRR measures the average reciprocal rank of predictions, with higher scores indicating more accurate predictions ranked closer to the top. Hits@K metrics assess whether the correct prediction is within the top K predictions, where Hits@1 is stricter than Hits@10 as it requires the correct prediction to be ranked first. These metrics serve to evaluate

TABLE I
Link prediction performance on FB15K-237 dataset.

	MRR	H@10	H@3	H@1
Our Framework	.3259	.5042	.3559	.2378
RGCN	.1948	.3647	.2073	.1135

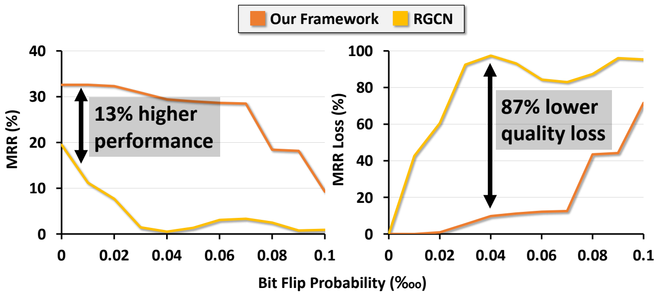

Fig. 8. Testing the performance of the different algorithms given different random bit flip probabilities. The x-axis is based on the per thousand range.

the performance of our model and the baseline in tasks such as link prediction and missing relationship inference within knowledge graphs. The results clearly demonstrate the superiority of our proposed approach, consistently surpassing the performance of the baseline model across a diverse range of evaluation metrics.

Moreover, to investigate the robustness of our framework under perturbations and uncertain conditions, we deliberately introduced noise to the weights of both the RGCN network and our proposed framework. As shown in Figure 8, our framework exhibited remarkable resilience to noise, consistently outperforming the baseline model under noisy conditions. This underscores the robustness and stability of our framework, validating its suitability for real-world applications where noise and uncertainties are pervasive.

Additionally, we conducted experiments to evaluate the performance of our framework considering the FeFET's non-idealities and different FFET device configurations.. Specifically, we employed graphs with varying edge counts of either 50 or 200, while maintaining a fixed number of 50 nodes. The hyperparameters α and η were set to 0.1, and the graph memory refinement used n as the number of iterations for the node memory reconstruction. These experiments

TABLE II

NOISE LEVELS FOR DIFFERENT THICKNESSES OF THE FERROELECTRIC LAYER, BIT PRECISION, AND TEMPERATURES. BASED ON [37].

HfO_2 Thickness (nm)	Precision (bit)	Front/Back Gate Read	Temepera- ture (°C)	Modeled Noise (%)
10	3	B	27	0.04
10	3	B	80	0.74
10	3	F	27	0.10
10	3	F	80	1.36
10	4	B	27	8.88
10	4	B	80	20.44
10	4	F	27	11.13
10	4	F	80	24.21
3	3	B	27	0.73
3	3	B	27	5.88
3	3	F	27	42.43
3	4	F	27	20.30
3	4	F	80	36.71

provided valuable insights into the trade-offs between bit-precision, dimensionality and performance of our framework under different scenarios, further substantiating the effectiveness and adaptability of our proposed approach.

B. Performance Evaluation of Framework: Bit Precision and Dimensionality Analysis in FeFET-based Graph Reasoning

In this section, we present the results of an extensive evaluation conducted to investigate the performance of our framework across various bit precisions (2-4 bits) and dimensionalities. The evaluation metric employed is Graph Reconstruction Accuracy (GRA), quantifying the percentage of nodes accurately retrieved its neighboring nodes from the memorized graph. The noise modeling and probabilities are applied to the graph memory hypervectors during inference.

For comparison, we establish a baseline configuration of a 10nm thick FE layer and 27 degrees Celsius. This configuration is chosen because it exhibits the least noise and a low probability impact within our framework, exhibiting a low probability of error of 0.10% for 3-bit precision and 11.13% for 4-bit precision. Additionally, two distinct configurations of FeFET cells, namely front gate and back gate, are considered in our study.

We explore various combinations of thickness (10 nm or 3 nm) and temperature (27 or 80 degrees Celsius) to comprehensively evaluate the framework's performance, as shown in Table II For the front gate configuration, as shown in Figure 9, with 3-bit precision and 50 edges, the most challenging scenario arises when operating at 3 nm thickness and 80 degrees Celsius, resulting in a substantial 42.43% probability of noise. To achieve performance comparable to the baseline in this challenging scenario, a minimum hypervector dimensionality of 7168 is required for both 3-bit and 4-bit precisions. Conversely, when dealing with 200 edges, the 3-bit precision falls short in achieving satisfactory performance, while the 4-bit precision necessitates a maximum dimensionality of 10240 to match the performance of the traditional von Neumann architecture under zero noise conditions.

In the case of the back gate configuration, the probabilities of error are reduced, with worst-case scenarios yielding 5.88% and 36.71% probabilities of error for 3-bit and 4-bit precisions, respectively. Notably, minimum hypervector dimensionality of 4096 is sufficient for all combinations involving 50 edges. With 200 edges, both 3-bit and 4-bit precisions exhibit satisfactory performance in the worst-case scenario as shown in Figure 10, requiring a minimum dimensionality of 7168 to achieve desirable results.

C. Hollistic noise exploration on our framework

In this experiment, we present a comprehensive analysis that explores the influence of bit-precision, noise probabilities of error, and dimensionality on the performance of our graph reasoning framework. In this analysis, the noise is not explicitly associated with specific configurations, allowing for a more generalized search. Noise probabilities ranging from 0% to 80% are considered, while dimensionalities span from 4012 to 10240.

Our framework is assessed under two noise introduction scenarios: during encoding and memorization of the graph and during inference. This evaluation sheds light on the framework's resilience to noise and capacity to adapt during learning.

Figure 11 presents the results for a graph with 50 edges with 2 bits where the model's performance falls short of traditional computing; however, it demonstrates an average improvement of 25% when noise is introduced during graph learning. Notably, our framework performs comparably to ideal conditions for the 3-bit and 4-bit scenarios, showcasing its robustness against noise. This observation underscores the framework's adaptability and ability to handle noise during the learning phase effectively.

In Figure 12, we shift our focus to a more challenging scenario involving 200 edges. In this case, both the 2-bit and 3-bit scenarios fail to achieve performance comparable to the ideal case. Nevertheless, when noise is introduced during training, substantial average improvements of up to 10.66% and 24.25% are observed for the 2-bit and 3-bit scenarios, respectively. Notably, for the 4-bit scenario, while the performance is not comparable when noise is introduced during inference, the introduction of noise during training yields highly promising results, with an average GRA of almost 90.3%.

These empirical findings highlight the robustness and adaptability of our framework in the presence of noise, particularly when noise is introduced during the learning phase. The results underscore the framework's potential to achieve high GRAs, even in challenging scenarios, positioning it as a viable solution for real-world applications requiring reliable and noise-tolerant graph reasoning capabilities.

D. Power and latency comparison

In this work, the power is modeled for the crossbar and the ADC. The contribution from the current mirror and the digital logic (inverters, adders) is negligible. The conductance values of the FeFETs and thus the power consumption is averaged for the crossbar (i.e., no data dependency). Additional digital logic such as an adder tree to combine the results from the individual crossbars into the full similarity and vector values is not considered. Similarly, the latency is based on electrical-level SPICE simulations for the crossbar and the ADC but not for the entire computing system.

Table V-D presents a comparative analysis across different platforms to highlight the advantages of the proposed framework with a CiM architecture. The original HDC graph reasoning model is implemented and tested on various existing CPU and GPU platforms, namely NVIDIA Jetson Orin SoC (referred to as "Jetson"), Intel i9-12900 CPU (referred to as "i9-12900"), and NVIDIA RTX 3090 GPU (referred to as "RTX 3090"). Additionally, we include the performance results of a previous study [44] that employed digital CiM acceleration, to

TABLE III

CROSS PLATFORM COMPARISON OF THE MEMORY HYPERVECTOR RECONSTRUCTION PROCESS. THE HYPERVECTOR DIMENSION IS 10000 AND 50 VERTICES ARE IN THE GRAPH.

	Jetson	i9-12900	RTX 3090	DCiM	ACiM 10nm*	ACiM 3nm*
Latency (ms)	702.5	237.5	9.25	17.75	15×10^{-6}	34×10^{-6}
Power (W)	60	53	61	6	0.297	0.171

* Only includes crossbars, not a complete computing system.

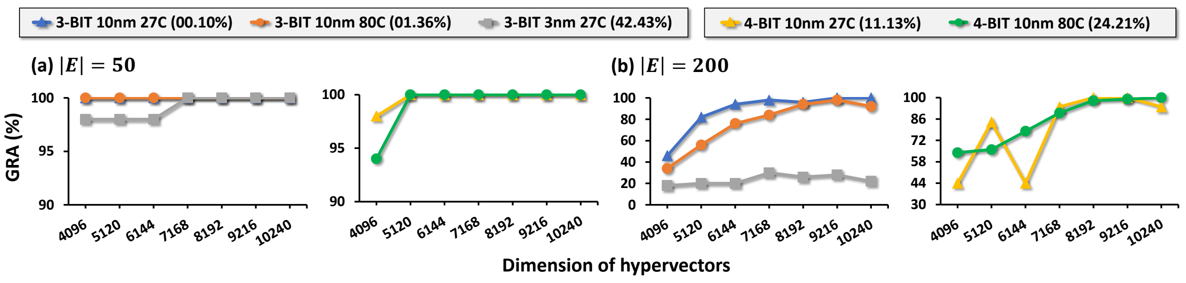

Fig. 9. Performance of proposed memory HDC learning of Graph HDC framework modeling the temperature for 10nm and 3nm thick on the Front gate FeFET configuration.

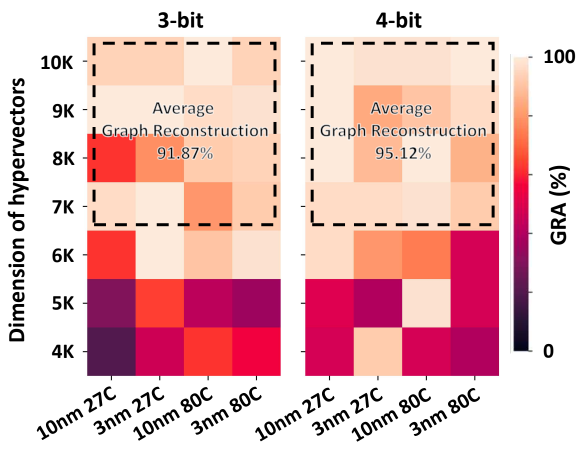

Fig. 10. Performance of proposed memory HDC learning of Graph HDC framework modeling the temperature for 10nm and 3nm thick on the Back gate FeFET configuration.

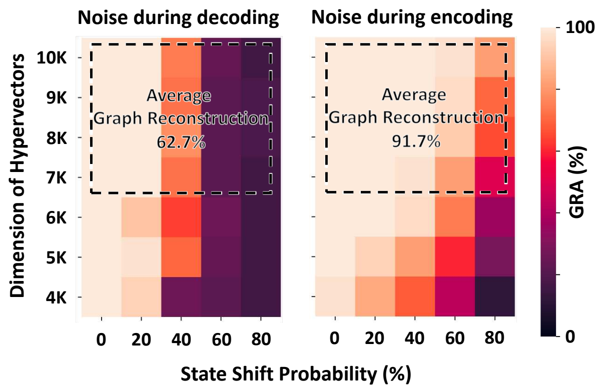

Fig. 11. Exploration of 2-bit proposed memory HDC learning of Graph HDC framework for different noise probabilities.

underscore the benefits of our proposed quantization method and analog CiM (ACiM). In this analysis, we assume all hypervectors dimension are 10000 and the graph size is 50 vertices and 200 edges. With our CiM architecture, all the data is stored in 8x16 FeFET-based crossbars enabling massive parallelism for the computations. Hence, the latency given in Table V-D is at nanosecond scale – two orders of magnitude faster than other platforms. Note that only the crossbar and ADC latency is modeled, modeling a complete computing system is deferred as part of future work. Nevertheless, the remaining digital logic is not expected to negate this advantage from the massive parallelism in the primary computations. The different in latency for the 10nm and 3nm FE layer thickness stems from the difference in current that the FeFETs conduct. At 3nm, the conductance is higher and thus the ladder ADC is slower, but consumes less power.

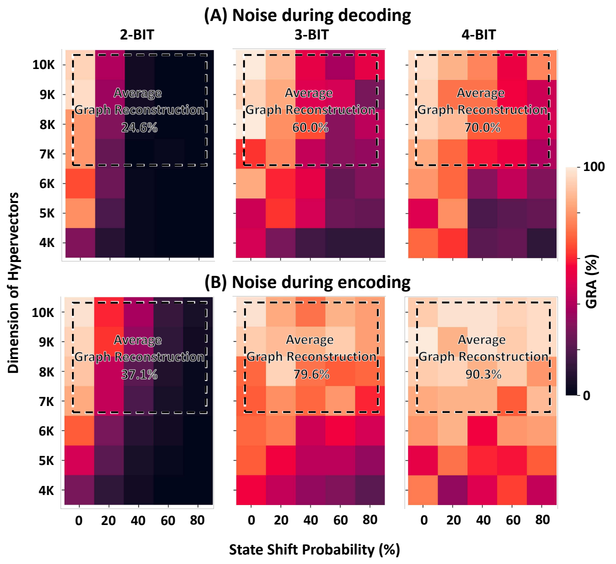

Fig. 12. Exploration of multi-bit proposed memory HDC learning of Graph HDC framework for different noise probabilities: (a) noise during decoding and (b) noise during encoding.

VI. Conclusions

Our study presents an innovative algorithm-hardware solution for efficient knowledge graph reasoning in AI systems. By harnessing hyperdimensional computing (HDC) and computing-in-memory (CiM) platforms, we overcome the memory wall problem and achieve robust computation. Through symbolic encoding and optimized information distribution, our HDC reasoning model enables parallel and fault-tolerant processing. The customized CiM architecture, based on FeFET technology, accurately represents the hardware platform. Our solution demonstrates comparable performance to traditional architectures, highlighting the potential of HDC and CiM for efficient AI reasoning.

VII. Acknowledgements

This work was supported in part by DARPA Young Faculty Award, National Science Foundation #2127780 and #2312517, and #2319198, Semiconductor Research Corporation (SRC), Office of Naval Research, grants #N00014-21-1-2225 and #N00014-22-1-2067, the Air Force Office of Scientific Research under award #FA9550-22-1-0253, by Advantest as part of the Graduate School "Intelligent Methods for Test and Reliability" (GS-IMTR) at the University of Stuttgart, and generous gifts from Xilinx and Cisco.

References

[1] N. Lao et al., "Random walk inference and learning in a large scale knowledge base," ser. EMNLP '11, Edinburgh, United Kingdom: Association for Computational Linguistics, 2011, pp. 529–539.

[2] A. Neelakantan et al., "Compositional vector space models for knowledge base completion," arXiv preprint arXiv:1504.06662, 2015.

[3] K. Liang et al., *Relational symmetry based knowledge graph contrastive learning*, 2022. arXiv: 2211.10738 [cs.AI].

[4] E. Shortliffe, "Computer-based medical consultations: Mycin," Artificial Intelligence - AI, vol. 388, 1976.

[5] S. Ji et al., "A survey on knowledge graphs: Representation, acquisition, and applications," IEEE Transactions on Neural Networks and Learning Systems, vol. 33, no. 2, pp. 494–514, 2022.

[6] C.-M. Wong et al., "Improving conversational recommender system by pretraining billion-scale knowledge graph," in *IEEE ICDE 2021*, 2021, pp. 2607–2612.

[7] M. Schlichtkrull et al., *Modeling relational data with graph convolutional networks*, 2017. arXiv: 1703.06103 [stat.ML].

[8] Y. Wang et al., "A gnn computing-in-memory macro and accelerator with analog-digital hybrid transformation and camenabled search-reduce," in *IEEE CICC 2023*, 2023, pp. 1–2.

[9] Y. Ni et al., "Neurally-inspired hyperdimensional classification for efficient and robust biosignal processing," in *Proceedings of the 41st IEEE/ACM International Conference on Computer-Aided Design*, 2022, pp. 1–9.

[10] N. McDonald et al., "Integrating complex valued hyperdimensional computing with modular artificial neural networks," in *Disruptive Technologies in Information Sciences VII*, SPIE, vol. 12542, 2023, pp. 152–170.

[11] N. McDonald, "Modularizing and assembling cognitive map learners via hyperdimensional computing," arXiv preprint arXiv:2304.04734, 2023.

[12] Z. Zou et al., "Scalable edge-based hyperdimensional learning system with brain-like neural adaptation," in *Proceedings of the International Conference for High Performance Computing, Networking, Storage and Analysis*, ser. SC '21, St. Louis, Missouri: Association for Computing Machinery, 2021.

[13] Z. Zou et al., "Memory-inspired spiking hyperdimensional network for robust online learning," Scientific Reports, vol. 12, no. 1, p. 7641, 2022.

[14] Z. Zou et al., "Eventhd: Robust and efficient hyperdimensional learning with neuromorphic sensor," Frontiers in Neuroscience, vol. 16, 2022.

[15] A. Hernandez-Cane et al., "Onlinehd: Robust, efficient, and single-pass online learning using hyperdimensional system," in *DATE*, IEEE, 2021, pp. 56–61.

[16] H. Chen et al., "Full stack parallel online hyperdimensional regression on fpga," in *IEEE ICCD 2022*, IEEE, 2022, pp. 517–524.

[17] H. Chen et al., "Darl: Distributed reconfigurable accelerator for hyperdimensional reinforcement learning," in *Proceedings of the 41st IEEE/ACM International Conference on Computer-Aided Design*, 2022, pp. 1–9.

[18] Y. Ni et al., "Algorithm-hardware co-design for efficient brain-inspired hyperdimensional learning on edge," in *2022 Design, Automation & Test in Europe Conference & Exhibition (DATE)*, 2022, pp. 292–297.

[19] Y. Ni et al., "Hdpg: Hyperdimensional policy-based reinforcement learning for continuous control," in *Proceedings of the 59th ACM/IEEE Design Automation Conference*, 2022, pp. 1141–1146.

[20] Z. Zou et al., "Biohd: An efficient genome sequence search platform using hyperdimensional memorization," in *Proceedings of the 49th Annual International Symposium on Computer Architecture*, 2022, pp. 656–669.

[21] S. Mittal, "A survey of reram-based architectures for processing-in-memory and neural networks," Machine Learning and Knowledge Extraction, vol. 1, no. 1, pp. 75–114, 2019.

[22] S. Tang et al., "Aepe: An area and power efficient rram crossbar-based accelerator for deep cnns," in *2017 IEEE 6th Non-Volatile Memory Systems and Applications Symposium (NVMSA)*, 2017, pp. 1–6.

[23] A. Kazemi et al., "Achieving software-equivalent accuracy for hyperdimensional computing with ferroelectric-based in-memory computing," Scientific Reports, vol. 12, no. 1, p. 19 201, 2022.

[24] H. Amrouch et al., "Brain-inspired hyperdimensional computing for ultra-efficient edge ai," in *2022 International Conference on Hardware/Software Codesign and System Synthesis (CODES+ISSS)*, 2022, pp. 25–34.

[25] H. E. Barkam et al., "Hdgim: Hyperdimensional genome sequence matching on unreliable highly scaled fefet," in *2023 Design, Automation & Test in Europe Conference & Exhibition (DATE)*, 2023, pp. 1–6.

[26] T. Geng et al., "I-GCN: A graph convolutional network accelerator with runtime locality enhancement through islandization," in *IEEE/ACM MICRO-54*, 2021, pp. 1051–1063.

[27] H. Zeng et al., "GraphACT: Accelerating GCN training on CPU-FPGA heterogeneous platforms," in *ACM/SIGDA FPGA 2020*, 2020, pp. 255–265.

[28] M. Yan et al., "HyGCN: A GCN accelerator with hybrid architecture," in *IEEE HPCA 2020*, IEEE, 2020, pp. 15–29.

[29] Y. Wang et al., "A gnn computing-in-memory macro and accelerator with analog-digital hybrid transformation and camenabled search-reduce," in *2023 IEEE Custom Integrated Circuits Conference (CICC)*, IEEE, 2023, pp. 1–2.

[30] I. Nunes et al., "Graphhd: Efficient graph classification using hyperdimensional computing," in *2022 Design, Automation & Test in Europe Conference & Exhibition (DATE)*, IEEE, 2022, pp. 1485–1490.

[31] J. Kang et al., "Relhd: A graph-based learning on fefet with hyperdimensional computing," in *2022 IEEE 40th International Conference on Computer Design (ICCD)*, IEEE, 2022, pp. 553–560.

[32] P. Poduval et al., "Graphd: Graph-based hyperdimensional memorization for brain-like cognitive learning," Frontiers in Neuroscience, p. 5, 2022.

[33] S. Dünkel et al., "A fefet based super-low-power ultra-fast embedded nvm technology for 22nm fdsoi and beyond," in *2017 IEEE International Electron Devices Meeting (IEDM)*, IEEE, 2017, pp. 19–7.

[34] K. Ni et al., "On the channel percolation in ferroelectric fet towards proper analog states engineering," in *2021 IEEE International Electron Devices Meeting (IEDM)*, 2021, pp. 15.3.1–15.3.4.

[35] P. R. Genssler et al., "On the reliability of fefet on-chip memory," IEEE Transactions on Computers, vol. 71, no. 4, pp. 947–958, 2021.

[36] H. Mulaosmanovic et al., "Ferroelectric field-effect transistors based on hfo2: A review," Nanotechnology, vol. 32, no. 50, p. 502 002, 2021.

[37] S. Chatterjee et al., "Comprehensive variability analysis in dual-port fefet for reliable multi-level-cell storage," IEEE TED, 2022.

[38] H. Mulaosmanovic et al., "Ferroelectric transistors with asymmetric double gate for memory window exceeding 12 v and disturb-free read," Nanoscale, vol. 13, no. 38, pp. 16 258–16 266, 2021.

[39] H.-K. Lim and J. G. Fossum, "Threshold voltage of thin-film silicon-on-insulator (soi) mosfet's," IEEE Transactions on electron devices, vol. 30, no. 10, pp. 1244–1251, 1983.

[40] E. P. Frady et al., "A theory of sequence indexing and working memory in recurrent neural networks," Neural Computation, 2018.

[41] S. Kullback et al., "On information and sufficiency," The annals of mathematical statistics, 1951.

[42] A. Gersho, "Asymptotically optimal block quantization," IEEE Transactions on information theory, 1979.

[43] P. L. Zador, Development and evaluation of procedures for quantizing multivariate distributions. Stanford University, 1964.

[44] R. M. Gray and D. L. Neuhoff, "Quantization," IEEE transactions on information theory, 1998.

[45] M. Imani et al., "Exploring hyperdimensional associative memory," in *IEEE HPCA*, IEEE, 2017, pp. 445–456.

[46] M. Imani et al., "Dual: Acceleration of clustering algorithms using digital-based processing in-memory," in *2020 53rd Annual IEEE/ACM International Symposium on Microarchitecture (MICRO)*, IEEE, 2020, pp. 356–371.

[47] S. Angizi et al., "Accelerating deep neural networks in processing-in-memory platforms: Analog or digital approach?" In *2019 IEEE Computer Society Annual Symposium on VLSI (ISVLSI)*, IEEE, 2019, pp. 197–202.

[48] M. Imani et al., "Floatpim: In-memory acceleration of deep neural network training with high precision," in *Proceedings of the 46th International Symposium on Computer Architecture*, 2019, pp. 802–815.

[49] S. Thomann et al., "Hw/sw co-design for reliable tcam-based in-memory brain-inspired hyperdimensional computing," IEEE Transactions on Computers, 2023.

[50] K. Monfaredi and H. Faraji Baghtash, "An extremely low-voltage and high-compliance current mirror," Circuits, Systems, and Signal Processing, vol. 39, no. 1, pp. 30–53, 2020.

[51] T. Soliman et al., "Felix: A ferroelectric fet based low power mixed-signal in-memory architecture for dnn acceleration," ACM Trans. Embed. Comput. Syst., vol. 21, no. 6, 2022.

[52] K. Toutanova and D. Chen, "Observed versus latent features for knowledge base and text inference," in *Proceedings of the 3rd workshop on continuous vector space models and their compositionality*, 2015, pp. 57–66.