HygHD: Hyperdimensional Hypergraph Learning

Jaeyoung Kang*, You Hak Lee*, Minxuan Zhou, Weihong Xu, and Tajana Rosing
University of California San Diego, USA
{j5kang, yhl004, miz087, wexu, tajana}@ucsd.edu

***Abstract*—Hypergraphs can model real-world data that has higher-order relationships. Graph neural network (GNN)-based solutions emerged as a hypergraph learning solution, but they face non-uniform memory accesses and accompany memory-intensive and compute-intensive operations, making the acceleration with near-data processing challenging. We propose a hyperdimensional computing (HDC)-based hypergraph learning framework called HygHD, which consists of highly parallelizable and lightweight HDC operations. HygHD accelerates both the training and inference on ferroelectric field-effect transistor (FeFET)-based processing-in-memory (PIM) hardware. Furthermore, we devise a hardware-friendly block-level concatenation and fine-grained block-level scheduler for high efficiency. Our evaluation results show that HygHD offers comparable accuracy to existing GNN-based solutions. Also, HygHD on GPU is up to $443\times$ ($7.67\times$) faster and $142\times$ ($2.78\times$) more energy efficient in training (inference) than the fastest GNN-based approach [1] on GPU. The HygHD accelerator further accelerates the HygHD algorithm, providing an average speedup of $40.0\times$ ($3.41\times$) on training (inference) compared to the HygHD GPU implementation.**

***Index Terms*—Hyperdimensional Computing, Hypergraph Learning, FeFET, Processing-in-memory**

I. Introduction

Hypergraphs can express complex higher-order entity relationships beyond traditional graphs. Unlike a graph that only considers pairwise relations between nodes using edges, an edge in hypergraph representation (hyperedge) can connect more than two nodes. Hypergraphs can capture and model many-to-many relationships. It has been used to model various real-world data, including co-citation/co-authorship relationships [1], social networks [2], and protein interactions [3].

Graph neural network (GNN) has become prevalent as it can analyze latent relationships between entities and perform machine learning (ML). Hypergraph learning has emerged because of its flexibility in the representation of complex relationships. However, hypergraph learning remains a challenging problem due to its complexity, e.g., Laplacian matrix computation [4]. Several graph convolutional network (GCN)-based methods have been proposed to represent a hypergraph in embedding space, such as HGNN [5], HyperGCN [1] that approximates hypergraph to utilize GCN and HNHN [6] that reflects nodes and hyperedges to the resulting representation through nonlinear activation in an iterative manner. However, these models are memory and compute-intensive during the forward pass and the back-propagation, respectively. Also, graph processing has highly random data access. Given its heterogeneous nature, the acceleration of hypergraph learning is challenging.

Besides, brain-inspired hyperdimensional computing (HDC) has shown its superior capability to embed the relationship between data with lightweight and parallelizable arithmetic operations. HDC models human cognition, which involves the simultaneous activity of a massive number of neurons, with a high-dimensional (HD) vector dubbed *hypervector* (HV). It enables memorization and association of information using element-wise addition and multiplication on HVs, respectively. Previously, [7], [8] redesigned graph ML problems with HDC operations. The end-to-end algorithm is memory-intensive, making the acceleration with processing in-memory (PIM) hardware feasible [8]. It offers scalable memory bandwidth and reduces data movement overhead. However, existing studies only work on simple graphs and are challenging to apply to real-world data like hypergraphs that have higher-order relations.

In this paper, we present an HDC-based hypergraph learning solution called HygHD. We show a novel and effective hypergraph HDC encoding strategy. In turn, we present how HygHD performs the training of the vertex classification of hypergraph learning only with element-wise vector addition. As HygHD is based on the HDC principle and trains without back-propagation, it is easily parallelizable and faster than existing GNN-based approaches [1], [5], [6]. Furthermore, we show that the HygHD algorithm is memory-intensive and accelerate it on PIM architecture to maximize efficiency. The main contributions of the paper are summarized as follows:

- We propose a hypergraph encoding strategy using the HDC concatenation. HygHD merges the information of member nodes into an HV for each hyperedge and preserves the similarity between a hyperedge and its members. The proposed algorithm trains the model without back-propagation and achieves comparable accuracy to existing solutions.
- To the best of our knowledge, this is the first hardware acceleration solution of hypergraph learning. We develop a ferroelectric field-effect transistor (FeFET)-based PIM accelerator, which accelerates hypergraph learning in-memory.
- We propose a block-level concatenation to enhance the accelerator efficiency. It avoids unnecessary read transactions and reduces peripheral circuits, yielding $1.31\times$ speedup on average compared to the naive concatenation.
- We present block-level scheduling for output HV generation. The scheduler is compatible with block-level concatenation and can flexibly assign blocks, yielding up to $1.24\times$ speedup over the HygHD with the existing cluster-level scheduler.

Our evaluation results show that HygHD is, on average, $566\times$ ($33.6\times$) faster and $231\times$ ($10.4\times$) more energy efficient on the training (inference) stage compared to the state-of-the-art GNN-based solutions [1], [5], [6] on GPU. Also, our HygHD PIM accelerator further improves speed and energy efficiency of training (inference) by $40\times$ ($3.41\times$) and $15395\times$ ($73003\times$) on average, respectively, over the GPU implementation.

*Equal contribution

979-8-3503-4859-0/DATE24/© 2024 EDAA

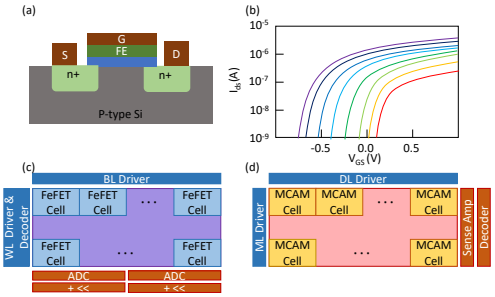


Fig. 1: (a) FeFET schematic (b) I-V characteristics of a FeFET cell [9] (c) Computation block schematic (d) Similarity block schematic

II. Background and Related Work

A. Hyperdimensional Computing Preliminaries

HDC represents data with *HV* and mimics the behavior of human memory by applying lightweight operations on HVs. **Reasoning** is done by measuring the similarity of two HVs, i.e., $\delta(\mathbf{H}_1, \mathbf{H}_2)$. Hamming distance or cosine similarity is used as a metric. **Bundling** mimics memorization, which is realized by element-wise vector addition. The resulting HV shows much higher similarity to operands than to a random HV. **Binding** associates two different pieces of information using Hadamard product. The resulting HV is orthogonal to operands. **Concatenation** ($\|$) combines multiple HVs into a single HV by extracting partial elements (equal size) from operands and concatenating them together. It effectively combines the operands' information with the same contribution. Consider the concatenation of three HVs, $\mathbf{A} = (a_0, \ldots, a_{D-1})$, $\mathbf{B} = (b_0, \ldots, b_{D-1})$, and $\mathbf{C} = (c_0, \ldots, c_{D-1})$, where D is the HV dimensionality. $\|(\mathbf{A}, \mathbf{B}, \mathbf{C})$ is the concatenation of $(a_0, \ldots, a_{\lfloor D/3 \rfloor})$, $(b_{\lfloor D/3 \rfloor+1}, \ldots, b_{\lfloor 2*D/3 \rfloor})$, and $(c_{\lfloor 2*D/3 \rfloor+1}, \ldots, c_{D-1})$.

B. Graph-based Machine Learning with HDC

The work in [7] encodes a graph to an HV and solves the graph classification with HDC operations and the PageRank algorithm. RelHD [8] generates an HV for each node that reflects neighbor information and node features and solves node classification using lightweight HDC operations. The authors accelerate the algorithm with a FeFET-based PIM accelerator that can efficiently handle in-memory computing on multi-bit HVs. However, these solutions are incapable of handling real-world data that has higher-order relations between entities, e.g., hypergraphs. Furthermore, the PIM accelerator in [8] is vulnerable to overflow as it uses binding operations and does not support concatenation operations that require more fine-grained control of FeFET memory blocks.

C. FeFET-based PIM

FeFET has emerged as a more suitable memory device for PIM because it is easier to integrate with CMOS, more scalable, and has better read/write energy efficiency than ReRAM [10]. The FeFET is a transistor integrating a Fe oxide layer into a gate dielectric stack of MOSFET [9] (see Fig. 1(a)). The ferroelectric oxide acts as an insulator. We can adjust the polarization of it with the gate voltage, and the threshold voltage (V_{th}) varies accordingly. We can represent 8 status (3 bits) in one cell by switching V_{th} [11] as shown in Fig. 1(b).

Previous studies have demonstrated the feasibility of performing addition and multiplication operations using FeFET [8], [11]. A multi-bit Content Addressable Memory (MCAM) has been proposed for HDC similarity search [8], [12]. In HygHD, a computation block utilizes FeFET-based addition, while a similarity block employs the FeFET MCAM for similarity checks (see Fig. 1(c),(d)).

III. HygHD Algorithm

We introduce the HygHD algorithm, a novel approach to mapping hypergraphs to HD space. Fig. 2(a) shows the flow of the HygHD algorithm. We show how node features and hyperedges are encoded into an HV and present HDC-based training and inference for hypergraph vertex classification.

A. Node Feature Encoding

In the node feature encoding stage, HygHD encodes the feature vector of each node to an HV called *node HV*, $\mathbf{N}$, with D dimensionality (see Fig. 2(b)). Different HDC encoding strategies can be used based on the feature vector characteristics. For dense feature vectors, position-level encoding [13] or non-linear encoding [14] can be used. For several datasets that use a bag-of-words (BoW) model, which represents the presence of a feature, i.e., $\{0, 1\}^F$. The BoW model can be encoded to a HV by (1) generating nearly orthogonal HVs $\mathbf{P}_{f_i} = \{-1, +1\}^D$ for feature f_i and (2) summing the generated HVs corresponding to feature indices that have non-zero values in the feature vector [8]. For example, in Fig. 2(b), node HV of v_1 ($\mathbf{N}_{v_1}$) is $\mathbf{P}_{f_0} + \mathbf{P}_{f_{F-2}}$ (orange boxes in Fig. 2(b)).

B. Hyperedge Encoding

We encode hyperedges to *hyperedge HVs*, $\mathbf{E}$. To encode a hyperedge into an HD space, we use the concatenation operation. Previous studies [7], [8] utilized binding operations to combine connected entities. However, the binding operation is unsuitable for capturing the unique characteristics of hyperedges. The binding operation combines two HVs with different characteristics to generate an orthogonal HV that possesses a new property. However, using the binding operation to embed hyperedges into the HD space is not effective because the resulting hyperedge HV would lack the individual characteristics of each connected node HV as they are all orthogonal to each other. Moreover, the binding operation increases the required bit-width per element of the HV if we use multi-bit HVs for high accuracy.

Fig. 2(c) shows the mapping of the hyperedge into HD space using concatenation operation. First, the number of nodes connected to a given hyperedge is determined. The HV of each node is then sliced into segments corresponding to the number of connected nodes in the hyperedge. Finally, relevant segments from the sliced node HVs are selectively combined to construct the HV representation of the hyperedge. In the example in Fig. 2(c), $\mathbf{E}_{e_1}$ can be computed with $\|(\mathbf{N}_{v_1}, \mathbf{N}_{v_2}, \mathbf{N}_{v_4})$.

C. Training the HDC model

Through the node feature and hyperedge encoding stages, we can represent node features and hyperedges in HVs. In the following, we show how HygHD solves vertex classification in a hypergraph, which has been tackled in existing works [1],

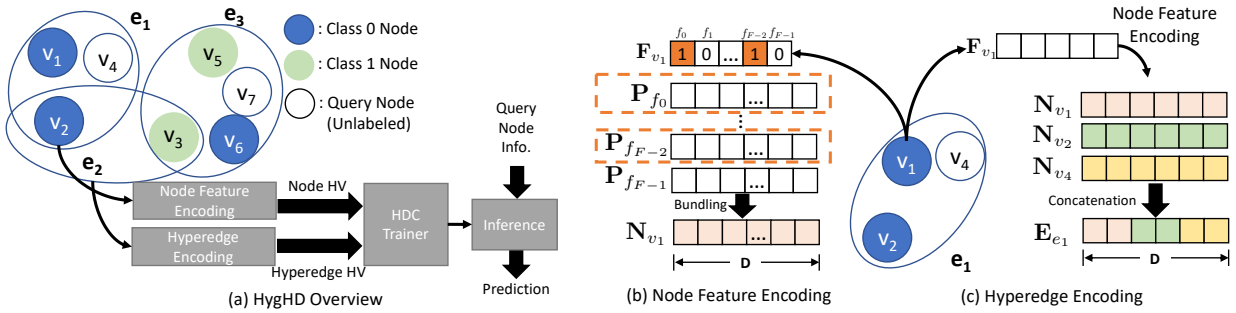


Fig. 2: HygHD algorithm overview.

[5], [6]. It aims to predict the classes of unlabeled nodes based on the topology, feature vectors of nodes, and labeled nodes.

During the training, we bundle node HVs ($\mathbf{N}$) and hyperedge HVs ($\mathbf{E}$) that belong to the same class. The HDC model memorizes and learns patterns from $\mathbf{N}$ and $\mathbf{E}$. First, we add the node HV and hyperedge HVs that are connected to each node; for node k, we compute the bundled HV $\mathbf{H}_k$ as $\mathbf{N}_k + \sum_{j \in B} \mathbf{E}_j$, where B is the set of indices for the hyperedges connected to node k. The representative HV for each class is generated by bundling the corresponding $\mathbf{H}$, i.e., $\mathbf{C}_i = \sum_{j \in L} \mathbf{H}_j$, where L is the set of indices of nodes labeled class i.

D. Inference

An unlabeled node in the test set has its node features and hyperedges. We build $\mathbf{H}$ for each query node using the same method used in the training. In turn, we measure the similarity between class HVs and $\mathbf{H}_q$ for the query node q. The class that shows the highest similarity is predicted as the label of a query. Our training strategy leverages the contributions of both the hyperedges and the node features. If a match is found, incorporating a part of the node's HV directly into the configuration of the hyperedge HV leads to a substantial increase in value during the similarity check.

IV. HygHD PIM Accelerator

The proposed algorithm can effectively speed up hypergraph learning due to the simplicity of HDC operations. Besides, this has a low operational intensity (≈1 ops/byte for each HygHD phase), implying that the algorithm is memory-intensive. FeFET-based PIM accelerators are state-of-the-art hardware for HDC [8], [11] that have shown high efficacy in various HDC operations on multi-bit HVs. However, the existing FeFET-based PIM accelerators for HDC cannot efficiently support HygHD algorithm due to the lack of support for concatenation operation as well as the incompatible scheduler. To this end, we devise a novel FeFET-based PIM acceleration that can effectively handle the HDC concatenation and develop a scheduler to maximize hardware utilization.

A. Hardware Architecture

Fig. 3 illustrates the architecture of HygHD PIM accelerator. Previous work [8] proposed a FeFET-based PIM accelerator for HDC algorithm on simple graphs. It organizes the memory space into a multi-level hierarchy consisting of 2D cell blocks, clusters, and tiles that process HDC operations with the row-wise FeFET operations, including both arithmetics and search

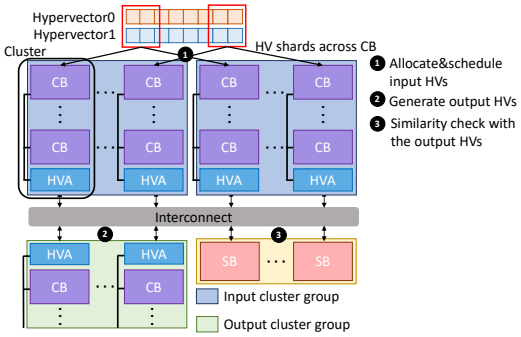


Fig. 3: HygHD accelerator design (CB: computation block, SB: similarity block, HVA: HV adder).

operations, as well as HV adders (HVAs), each attached to a cluster of blocks. However, the previous hardware does not support the HDC concatenation. If the concatenation is implemented naively, it requires additional circuits for component (element)-level control of HVs. In addition, existing schedulers schedule operand blocks at the cluster level, which can cause longer critical paths due to limited parallelism. To this end, we propose a *block-level FeFET-based PIM*. Our design enables more efficient computation of operations specific to HygHD and reduces latencies through fine-grained block-level scheduling. The blocks in HygHD accelerator consist of computation blocks and similarity blocks. The similarity block is utilized for similarity check during the inference, while all other operations are carried out within the computation block.

B. Dataflow

HygHD hardware divides the memory space into input clusters and output clusters since all stages of HygHD follow a similar dataflow pattern. The output cluster group stores a single copy of the result of the current stage, while the input cluster group stores multiple replicates of input data to maximize parallelization. Therefore, the size of the output cluster group (i.e., the number of allocated blocks) depends on the number of HVs generated in the stage, while the size of the input cluster group is determined by the product of the number of input HVs and the computation parallelism. Considering the dimension of the HV exceeds the column size of a single block in general, HVs are divided and distributed across several blocks, and elements of different HVs are aligned in the same column inside clusters. After the allocation, the operations are executed according to the predetermined schedule for each process.

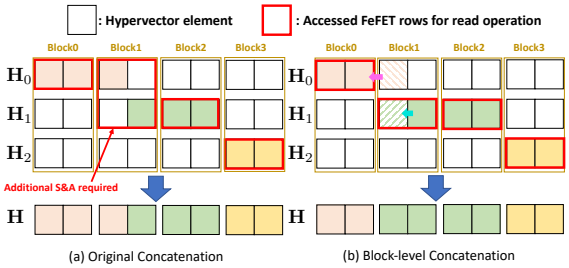


Fig. 4: An example of block-level HDC concatenation.

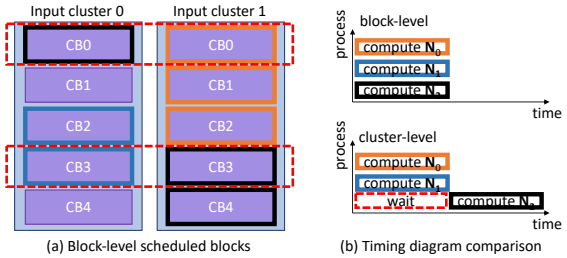


Fig. 5: The proposed block-level scheduler.

The input clusters compute various HD operations, including element-wise arithmetic and concatenation using a combination of FeFET blocks and HVAs. The computation results are sent to the output cluster via the interconnect network.

C. Block-level Concatenation

HygHD uses the concatenation to represent a hyperedge. The HV of each node is sliced into segments corresponding to the number of nodes connected by the edge, and these segments are then combined to form the HV of the hyperedge. A naive approach for implementing the concatenation operation ($\mathbf{H} = \|(\mathbf{H}_1, \mathbf{H}_2, \mathbf{H}_3)$) in hardware is illustrated in Fig. 4(a)). The relevant segments of the HV for each node are read at block levels. Subsequently, these segments are connected to the sliced HV of the other node through the shift-and-add operation and then written to the output HV (i.e., the HV of the hyperedge). However, implementing the HDC concatenation in hardware is not well-suited for the PIM structure that processes data at a block level. This is due to the additional requirements of read and add&shift operations, which cause inefficiencies in terms of both hardware utilization and processing speed.

We propose a hardware-friendly block-level HDC concatenation. As depicted in Fig. 4(b), the block-level concatenation involves slicing the HV of the node at the block level and connecting these blocks to create the HV of the hyperedge. We can eliminate additional read transactions and add&shift operations. Due to the fact that the dimension of the HV is typically over 4000 while the block size is usually 64, implementing the hyperedge HV using block-level concatenation has minimal impact on accuracy.

D. Block-level Scheduler

Efficient scheduling plays a significant role in optimizing hardware performance in the HygHD accelerator. When multiple output HVs require input HVs from the same block, the operations need to run in serial, leading to increased latencies. By scheduling internal operations in HygHD stages in advance with the proper order, we can improve overall efficiency.

Algorithm 1 Block-level scheduling

```
1:  Goal: Set a scheduling dictionary SD (keys: output HVs; values: block-
    cluster dictionaries BCD (keys: input blocks; values: input clusters))
2:  Initialize SD
3:  Generate a dictionary OID (keys: output HVs; values: input blocks)
4:  for N_k in (output HVs) do
5:      input blocks ← OID[N_k]
6:      Initialize BCD
7:      for CB_i in (input blocks) do
8:          for cluster in (input clusters) do
9:              Find the cluster (target_cl) that has the least usage of CB_i.
10:         end for
11:         Update BCD with {CB_i: target_cl}
12:     end for
13:     Update SD with {N_k: BCD}
14: end for
15: return SD
```

The existing scheduling method [8] is not suitable for HygHD. This is because it tries to assign all input blocks to a single cluster based on the output HV, which can lead to suboptimal scheduling decisions in HygHD. Let us consider a scenario with only two input clusters (input cluster 0 and 1) shown in Fig. 5(a). Both input clusters contain a copy of computation blocks (CB0-CB5). When output HV $\mathbf{N}_0$ requires CB0-2, the cluster-level scheduler assigns these computation blocks within the same cluster (input cluster 1 in Fig. 5(a)). Besides, when output HV $\mathbf{N}_1$ requires CB2-3, the cluster-level scheduler realizes that CB2 overlaps in the input cluster 1 and decides to perform this operation on the other cluster (input cluster 0). At the same time, if output HV $\mathbf{N}_2$ also requires CB0 and CB3-4, we cannot proceed with the operation for $\mathbf{N}_2$ in parallel with the cluster-level scheduling.

We propose a block-level scheduler shown in Algorithm 1. This approach aims to allocate blocks to clusters in a way that minimizes the total latency across all blocks (L7, L8) in a greedy manner. It schedules each input CB, searching the cluster that has the least usage of the input CB (L9). For example, in the same case with the cluster-level scheduling (Fig. 5(a)), CB0 can be allocated to cluster 0, while CB2-3 can be assigned to cluster 1 for $\mathbf{N}_2$. Consequently, $\mathbf{N}_0$, $\mathbf{N}_1$, $\mathbf{N}_2$ can be computed in parallel (Fig. 5(b)).

The block-level control offers several advantages: more parallelism, elimination of the need for sorting output vectors for scheduling order, and improved compatibility with block-level concatenation. Note that blocks involved in the operation of a particular output HV can be assigned to multiple clusters.

V. Evaluation

A. Experimental Setup

System Environment: We implemented the HygHD on NVIDIA RTX 4090 and used `nvidia-smi` to measure power consumption. HygHD on GPU adopts the state-of-the-art GPU optimizations for HDC [15], [16]. For the HygHD accelerator evaluation, we integrate latency and energy consumption estimates to a PyTorch Geometric-based simulator. We use the FeFET device with 45nm technology in [8], [11], [17], [18], and estimate latency and energy consumption in FeFET blocks and peripheral circuits such as shift-and-add and ADCs using NeuroSim [19]. We synthesized the HVA with Synopsys

TABLE I: System specifications of HygHD.

HygHD	
Blocks	64×64 3 FeFET cells, 8 columns/ADC, Add&Shift
Clusters	1024 blocks, HVA, 8KB SRAM
Tiles	128 clusters, interconnect bandwidth = 16GB/s
Computation Blocks	
tADD=86.8ns, tRD=42ns, tWR=82ns eADD=5.725pJ, eRD=0.483pJ, eWR=0.618pJ	
FeFET CAM Cells in Similarity Blocks	
tSearch=1.5ns (64bit), eSearch=0.069fJ/bit	
Area and Power of Components	
FeFET blocks	1.26mm^2/460.8mW
ADC	0.93mm^2/1.54W
DAC	0.066mm^2/1.54W
Add & Shift	0.36mm^2/307.2mW
HVA & SRAM	0.057mm^2/27.34mW

TABLE II: Dataset attributes

	Co-citation			Co-authorship		Text
	Cora (CR-C)	CiteSeer (CS-C)	Pubmed (PB-C)	Cora (CR-A)	DBLP (DBLP-A)	20News (20N-U)
$\lvert\mathcal{V}\rvert$	2708	3312	19717	2708	43413	16242
$\lvert\mathcal{E}\rvert$	1579	1079	7963	1072	22535	100
$\lvert\mathcal{F}\rvert$	1433	3703	500	1433	1425	100
$\lvert\mathcal{C}\rvert$	7	6	3	7	6	4
max $\lvert e\rvert$	5	26	171	43	20	2241

Design Compiler 65nm library and scaled it to 45nm. The design was placed and routed using Synopsys IC Compiler.

Baselines and Datasets: We compare HygHD to GCN-based hypergraph learning solutions, HyperGCN [1], HNHN [6], and HGNN [5], in terms of accuracy, speed, and energy efficiency. We enable mediators and fast mode (known as FastHyperGCN) for HyperGCN. HNHN evaluation and HGNN evaluation were performed on hypergraph benchmark tool [20]. Baselines are trained for 200 epochs, which is required for peak accuracy. We use real-world datasets [1], [21]: co-citation Cora (CR-C), CiteSeer (CS-C), and Pubmed (PB-C); co-authorship Cora (CR-A) and DBLP (DBLP-A); 20Newsgroup (20N-U) (details in Table II). $|\mathcal{V}|$, $|\mathcal{E}|$, $|\mathcal{F}|$, and $|\mathcal{C}|$ denote the number of vertices, hyperedges, features, and classes, respectively. "max $|e|$" is the maximum number of nodes each hyperedge includes. Note that we only compare HygHD and baselines on GPU since there has been no accelerator for hypergraph learning.

HygHD and PIM Configurations: We set HV dimensionality to 8192. Operand HVs are duplicated to enhance parallelism in our accelerator. Our PIM architecture has the capability to accommodate a maximum of 8 copies. In order to achieve our desired parallelism, we need 2, 2, 4, 2, 8, and 4 tiles for CR-C, CS-C, PB-C, CR-A, DBLP-A, and 20N-U, respectively.

B. Performance Comparison to GNN-based Solutions

We perform speedup and energy efficiency comparisons between our HygHD and baselines on GPU. Fig. 6 shows the speedup and energy efficiency improvement of HygHD for training and inference phase over baselines. HygHD is 566× and 33.6× faster on average for training and inference than baselines, respectively. Also, HygHD is, on average, 231× and 10.4× more energy efficient on training and inference, respectively, than others. For all cases, HygHD consumes more power than baselines as it can accommodate more parallelism through element-wise vector operations. However, the significant reduction in execution time compensates for the increased power consumption and enhances energy efficiency.

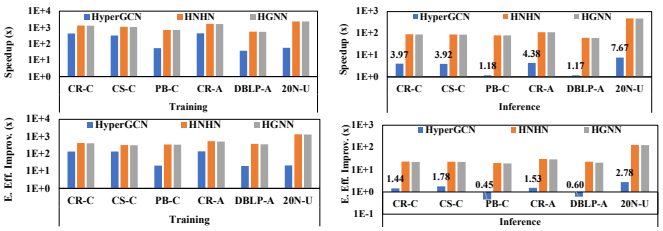


Fig. 6: Speed and energy efficiency of HygHD vs. baselines on GPU

TABLE III: Accuracy comparison with GNN-based methods

Accuracy (%)	CR-C	CS-C	PB-C	CR-A	DBLP-A	20N-U
HyperGCN	**0.68**	0.63	0.74	0.7	0.76	**0.81**
HNHN	0.65	0.62	**0.77**	0.63	0.86	**0.81**
HGNN	0.66	**0.67**	**0.77**	0.69	**0.88**	0.80
HygHD (Ours)	**0.68**	0.63	**0.77**	**0.72**	0.86	0.76

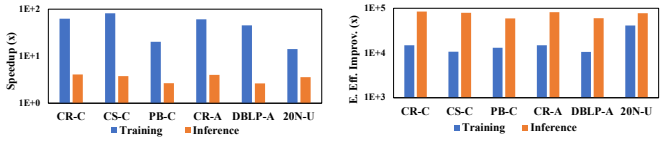
For the training phase comparison, we included node feature encoding and hyperedge encoding stages for HygHD. The training time for baselines includes Laplacian operations. HygHD shows lower latency with lower energy consumption across all datasets thanks to lightweight HDC operations. HygHD enables the training with element-wise addition. In contrast, GNN-based baselines accompany complex computations like matrix multiplication, Laplacian, and back-propagation. These operations used for GNN-based methods consume more time and energy. Moreover, while HygHD needs only single-pass training, all baselines require iterations (at least a few hundred epochs) in training to achieve peak accuracy.

The inference of HygHD also shows lower latency and energy consumption compared to baselines in most cases. Only in PB-C and DBLP-A datasets, HyperGCN shows 0.45× and 0.6× lower energy consumption than HygHD. This is because HyperGCN converts a hypergraph to a conventional graph during the training phase and processes GCN over it. The inference step reuses the hypergraph approximation. However, in PB-C and DBLP-A, HyperGCN shows the lowest accuracy among the others. In the DBLP-A dataset, HyperGCN results in more than a 12%p accuracy drop compared to state-of-the-art accuracy in the DBLP-A dataset as shown in Table III.

The dimensionality of HVs (D) in HygHD has an impact on various factors such as accuracy, execution time, and hardware size. When D is low, the accuracy drops as it limits the amount of expressed and compressed information, but the hardware size and the required number of computations are small. On the other hand, high dimensionality can enhance accuracy, but the computational complexity and size of the hardware increase. When $D = 8192$, HygHD achieves peak accuracy.

C. HygHD on PIM vs. HygHD on GPU

Fig. 7 shows a comparison between the HygHD PIM accelerator and the HygHD GPU implementation in terms of their speed and energy efficiency. The proposed accelerator performs on average 40× and 3.41× faster in training and inference, respectively, compared to HygHD on GPU. Furthermore, HygHD on PIM is 15395× and 73003× more energy efficient on average for training and inference, respectively, than HygHD on GPU. HygHD uses specialized blocks: computation and similarity blocks for the training and inference stages, respectively. These blocks are optimized for HDC operations



(a) Speed (b) Energy efficiency

Fig. 7: HygHD on PIM vs. HygHD on GPU

(a) (b)

Fig. 8: HygHD on accelerator analysis (a) Latency breakdown of HygHD on GPU and PIM (b) Scalability of HygHD

like bundling, concatenation, and similarity check (reasoning). Also, with proper scheduling, we can efficiently parallelize them. From an energy consumption perspective, GPUs are known for high power consumption, making the large energy efficiency gap between HygHD on PIM and GPU.

D. HygHD PIM Accelerator Analysis

Breakdown. Fig. 8(a) shows the latency breakdown of HygHD on GPU and PIM. Compared to GPU, PIM handles hyperedge encoding and training phases well, and the majority of latency concentrates on node encoding. Even though there are multiple copies (up to 8) of input operands and the scheduler controls the execution order, the same input HV can be used a maximum of 8 times at the same time. Besides, the node encoding phase deals with more output HVs than their inputs (or operands), leading to increased latency. For the PB-C case, the number of features, which is equivalent to the number of operands, is only 500. However, the number of vertices, which corresponds to the output HVs, is significantly larger ($19717/500 = 39\times$).

Cluster-level vs Block-level Scheduling. In HygHD, we use a block-level scheduler, which efficiently manages resources and parallelizes operations with fine-grained control of FeFET memory blocks. Compared to the previous cluster-level scheduler [8], the block-level scheduler in HygHD enhances the speed by $1.24\times$ speedup on average across datasets.

Scalability. Processing larger hypergraphs requires higher memory bandwidth and capacity. The proposed accelerator can scale them by increasing CB and SB. We compare the runtime according to the number of CB and SB. We set the number of blocks in the multiple of minimum required blocks ($2\times$, $4\times$, and $8\times$). As shown in Fig. 8(b), the degree of speedup increases as we increase CB and SB: $10.1\times$ on average in an $8\times$-scale scenario. It implies that our accelerator offers scalable memory bandwidth and can easily support larger hypergraphs.

Effectiveness of Block-level Concatenation. The proposed block-level concatenation allows HygHD to remove unnecessary read transactions and add&shifts and implement concatenation operations with only reads and writes. Our experiment data shows that block-level concatenation is $1.31\times$ faster compared to the naive approach across all datasets on average.

VI. Conclusion

We presented HygHD, an HDC-based hypergraph learning framework. HygHD represents hyperedges as HVs using concatenation and trains the model with only element-wise vector additions. Furthermore, we developed a FeFET-based PIM accelerator to accelerate the proposed algorithm. We introduced hardware-friendly block-level concatenation to reduce peripheral circuits. Also, we devised a block-level scheduler to efficiently handle operations in the HygHD algorithm through fine-grained control of FeFET memory blocks. Our evaluation shows that HygHD runs up to $443\times$ ($7.67\times$) faster with $142\times$ ($2.78\times$) improved energy efficiency in the training (inference) stage over the fastest GNN-based hypergraph solution [1] on GPU while showing comparable accuracy. Furthermore, our accelerator offers, on average, $40.0\times$ ($3.41\times$) speedup and $15395\times$ ($73003\times$) energy efficiency improvement on the training (inference) stage compared to HygHD on GPUs.

Acknowledgment

This work was supported in part by PRISM and CoCoSys, centers in JUMP 2.0, an SRC program sponsored by DARPA, SRC Global Research Collaboration (GRC) grants, and NSF grants #1826967, #1911095, #2003279, #2052809, #2112665, #2112167, #2100237, and #2023-JU-3135.

References

[1] N. Yadati *et al.*, "Hypergcn: A new method for training graph convolutional networks on hypergraphs," *NeurIPS*, 2019.
[2] D. Arya *et al.*, "Exploiting relational information in social networks using geometric deep learning on hypergraphs," in *ICMR*, 2018.
[3] K. A. Murgas *et al.*, "Hypergraph geometry reflects higher-order dynamics in protein interaction networks," *Scientific Reports*, vol. 12, 2022.
[4] Y. Gao *et al.*, "Hypergraph learning: Methods and practices," *IEEE TPAMI*, vol. 44, no. 5, pp. 2548–2566, 2020.
[5] Y. Feng *et al.*, "Hypergraph neural networks," in *AAAI*, 2019.
[6] Y. Dong *et al.*, "Hnhn: Hypergraph networks with hyperedge neurons," *arXiv preprint arXiv:2006.12278*, 2020.
[7] I. Nunes *et al.*, "Graphhd: Efficient graph classification using hyperdimensional computing," in *DATE*, 2022.
[8] J. Kang *et al.*, "Relhd: A graph-based learning on fefet with hyperdimensional computing," in *ICCD*. IEEE, 2022.
[9] K. Ni *et al.*, "A circuit compatible accurate compact model for ferroelectric-fets," in *VLSIT*, 2018.
[10] Y. Long *et al.*, "A ferroelectric fet-based processing-in-memory architecture for dnn acceleration," *JXCDC*, 2019.
[11] A. Kazemi *et al.*, "Mimhd: Accurate and efficient hyperdimensional inference using multi-bit in-memory computing," in *ISLPED*, 2021.
[12] ——, "In-memory nearest neighbor search with fefet multi-bit content-addressable memories," in *DATE*, 2021.
[13] M. Imani *et al.*, "Voicehd: Hyperdimensional computing for efficient speech recognition," in *ICRC*, 2017.
[14] ——, "DUAL: Acceleration of clustering algorithms using digital-based processing in-memory," in *MICRO*, 2020.
[15] J. Kang *et al.*, "Xcelhd: An efficient gpu-powered hyperdimensional computing with parallelized training," in *ASP-DAC*, 2022.
[16] ——, "Openhd: A gpu-powered framework for hyperdimensional computing," *IEEE TC*, 2022.
[17] A. Kazemi *et al.*, "In-memory nearest neighbor search with fefet multi-bit content-addressable memories," in *DATE*, 2021.
[18] X. Yin *et al.*, "Fecam: A universal compact digital and analog content addressable memory using ferroelectric," *IEEE T-ED*, 2020.
[19] X. Peng *et al.*, "Dnn+neurosim: An end-to-end benchmarking framework for compute-in-memory accelerators with versatile device technologies," in *IEDM*, 2019.
[20] H. Hwang *et al.*, "Hyfer: A framework for making hypergraph learning easy, scalable and benchmarkable," in *WWW W' on Graph Learning Benchmarks*, 2021.
[21] D. Dua *et al.*, "UCI machine learning repository," 2017. [Online]. Available: http://archive.ics.uci.edu/ml