

# Grin: HyperGNN Training Framework for Efficient Edge Inference via Hypergraph Restructuring

Chaofang Ma<sup>†</sup>, Lin Jiang<sup>‡</sup>, Zeyu Li<sup>†</sup>, Xingyu Liu<sup>†</sup>, Jiang Xu<sup>§</sup>, and Wei Zhang<sup>†\*</sup>

<sup>†</sup>The Hong Kong University of Science and Technology, <sup>‡</sup>Northeastern University,

<sup>§</sup>The Hong Kong University of Science and Technology (GZ)

\*Corresponding author: wei.zhang@ust.hk

**Abstract**—Hypergraph neural networks (HyperGNNs) have garnered increasing attention for their ability to model high-order relationships in various domains. However, the extremely sparse connections inherent to hypergraphs result in numerous off-chip memory accesses, posing a long-latency inference issue on edge devices. Existing hardware accelerators focus solely on exploiting the limited data reuse opportunities in hypergraphs to mitigate this issue, without addressing the underlying cause: the sparsity of the hypergraph structures themselves.

To address the fundamental limitation, this paper proposes Grin, a general HyperGNN training framework. It is designed to restructure hypergraphs for enhancing inference efficiency on edge devices regardless of hardware architectures while improving model performance. Specifically, hyperedge pruning within Grin is utilized to eliminate redundant computation workloads, effectively lowering overall off-chip memory accesses. Moreover, Grin redefines the objective of traditional data augmentation by incorporating hardware efficiency alongside model accuracy. This shift enables significantly increased data reuse in the remaining computation workloads, thereby ensuring model performance and further reducing off-chip memory accesses. Experiments demonstrate that, with increased model accuracy, deploying Grin-optimized hypergraphs on the state-of-the-art (SOTA) accelerator achieves an average inference speedup of 1.41× compared to the original hypergraphs on the same accelerator, while reducing off-chip memory accesses by 27.60%. Furthermore, this deployment achieves a 14.82× speedup over the SOTA GPU-based system.

**Index Terms**—hypergraph neural network, training framework, edge inference, data augmentation, data reuse.

## I. INTRODUCTION

Graph neural networks (GNNs) have evolved into a basic approach for machine learning tasks [1]–[4]. A notable characteristic of GNNs is their reliance on graphs for training, where relationships among entities are represented through pairwise connections. However, in real-world scenarios, interactions among entities are more complex and cannot be adequately described by pairwise relationships alone. Therefore, GNNs struggle to capture high-order dependencies, which constrains their effectiveness across diverse application domains. To overcome the limitation, hypergraph neural networks (HyperGNNs) have been introduced [5]–[7]. By leveraging hypergraphs, where each hyperedge can simultaneously connect multiple nodes, HyperGNNs model intricate data correlations more efficiently. Empirical studies have indicated that HyperGNNs outperform GNNs, leading to their adoption in various fields, such as action recognition [8], recommendation [9], [10], biology system modeling [11], and Alzheimer’s disease detection [12].

However, the enhanced applicability of HyperGNNs comes at the expense of more complex computation patterns, leading to the long-latency issue in inference, especially on resource-constrained edge devices. The key contributor to the inefficiency is the time-consuming two-stage aggregation phase in HyperGNNs: messages are first propagated from nodes to their connected hyperedges, and then returned to the nodes. Since real-world hypergraphs are extremely sparse, this aggregation phase triggers numerous irregular memory usage demands, leading to frequent off-chip memory accesses and significantly increased inference latency. As a result, the aggregation phase becomes the performance bottleneck in inference. The profiling results in [13] further corroborate this conclusion, revealing that the aggregation phase accounts for nearly 92.7% of the total latency on an NVIDIA A100 GPU.

Minimizing off-chip memory accesses is essential for efficient HyperGNN inference on edge devices. Existing HyperGNN accelerators focus on hardware optimizations to exploit available data reuse opportunities for reducing such accesses. For instance, RAHP [13] presents a dedicated data loader that schedules tasks based on the principle of maximizing data reuse, thereby reducing off-chip memory accesses. MeHyper [14] develops an accelerator to eliminate redundant computations in the aggregation phase by leveraging data reuse opportunities, effectively reducing duplicate intermediate calculations and associated memory traffic. However, the underlying cause of the inefficient inference issue lies in the extreme sparsity of hypergraphs, which inherently offer few opportunities for data reuse. Hence, hardware optimizations yield only marginal performance gains, without touching the fundamental cause.

In contrast, some software methods that optimize hypergraph structures solely for model performance offer promising potential to address the underlying cause of inference inefficiency. Among these, data augmentation stands out as a particularly compelling direction. Without additional data collection, it introduces diversity during training by altering the original data, including features, labels, and graph or hypergraph structures, to improve model performance (e.g., accuracy) [15]–[17]. Inspired by structure manipulation methods in data augmentation that add or remove edges or hyperedges to introduce diversity, we identify crucial potential in leveraging hypergraph restructuring to improve data reuse for efficient inference while preserving model performance. However, existing data augmentation methods focus solely on enhancing model performance,

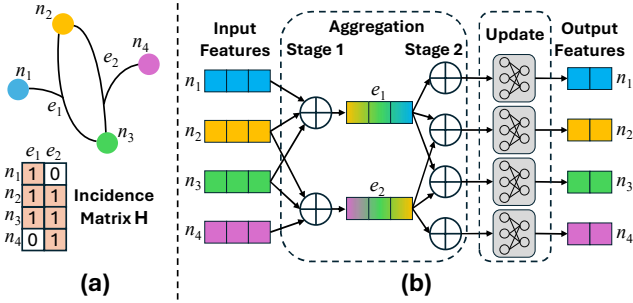


Fig. 1. Hypergraph and HyperGNN: (a) A hypergraph and its incidence matrix; (b) HyperGNN inference paradigm within each layer.

limiting their capacity to achieve this dual goal. To bridge the gap, we introduce **Grin**, a HyperGNN training framework that restructures hypergraphs to enable efficient edge inference. It is compatible with existing HyperGNN accelerators, regardless of hardware architectures. To the best of our knowledge, Grin is the first work to extend traditional data augmentation methods for accelerating inference. Our contributions are as follows:

- We propose Grin, a general HyperGNN training framework that restructures hypergraphs for efficient edge inference regardless of hardware architectures, while also improving model performance.
- Within Grin, we present a hardware-friendly augmentation method that optimizes hypergraph structures in a fine-grained manner, further enhancing data reuse.
- Experiments show that, with increased model accuracy, deploying Grin-optimized hypergraphs on the state-of-the-art (SOTA) accelerator yields inference speedups of 1.41 $\times$  and 14.82 $\times$  against the original hypergraphs on the same accelerator and the SOTA GPU-based system, respectively.

## II. BACKGROUND AND MOTIVATION

### A. Hypergraphs and HyperGNNs

A hypergraph is defined as  $HG = (N, E)$ , where  $N = (n_1, n_2, \dots, n_{|N|})$  and  $E = (e_1, e_2, \dots, e_{|E|})$  denote the node list and hyperedge list, respectively. The distinctive property of a hypergraph against a graph is that each hyperedge can simultaneously connect multiple nodes. To represent the relationships between hyperedges and nodes, an incidence matrix  $\mathbf{H}$  of size  $|N| \times |E|$  is employed. In this matrix, an element of 1 indicates that a node is connected to a hyperedge, whereas an element of 0 indicates no connection. For example, in Fig. 1(a),  $e_1$  connects three nodes:  $n_1$ ,  $n_2$ , and  $n_3$ . Hence,  $\mathbf{H}[1, 1]$ ,  $\mathbf{H}[2, 1]$ , and  $\mathbf{H}[3, 1]$  are set to 1 to reflect these connections, while  $\mathbf{H}[4, 1]$  remains 0 to indicate the absence of any connection.

HyperGNNs are designed to perform machine learning tasks on hypergraphs. Representative models include HGNN [18], UniGIN [19], and HGNN+ [20]. While these models differ in architectural details, they follow a unified inference paradigm within each layer. As illustrated in Fig. 1(b), the overall process comprises two phases: aggregation and update. Generally, the aggregation phase consists of two stages. Stage 1 involves transmitting node features to their connected hyperedges, where each hyperedge aggregates the incoming messages to construct its representation. In Stage 2, these hyperedge representations

are then propagated to the associated nodes, which in turn aggregate the received messages to update their features. Finally, in the update phase, each node refines its feature representation by applying a linear transformation to the aggregated messages.

### B. Data Augmentation

During the training of GNNs, data augmentation methods aim to enhance the existing data, rather than relying on additional data collection or labeling, to improve model performance [15]–[17]. To this end, these methods typically introduce variations to the original data, such as node features and graph structures, thereby enriching data diversity. Among various data augmentation methods, edge manipulation stands out as a particularly effective approach. Specifically, it alters the graph topology by removing existing edges and adding potential ones that do not exist originally according to specific rules [21]–[23]. This process introduces structural variations that prevent overfitting while preserving the essential information of the original graphs (note that specific data augmentation methods for hypergraphs exist [24]–[26], although they are not discussed here, as they fall outside the scope of this paper).

To effectively determine which edges to eliminate or add, the method proposed in [23] utilizes the graph auto-encoder (GAE) [27] as an edge predictor. Specifically, GAE adopts a simple yet effective structure: a two-layer graph convolutional network model [1] as the encoder, paired with an inner-product-based decoder. It takes node features and original edges as input and outputs an edge importance matrix  $\mathbf{M}$  of size  $|N| \times |N|$ . Each element in this matrix, ranging from 0 to 1, represents the importance of the corresponding node-to-node connection for enhancing model performance. The edge importance matrix then guides edge manipulation in [23]: edges with low importance are removed, whereas high-importance edges that are not part of the original graph structures are introduced.

However, these augmentation methods focus solely on improving model performance, overlooking the inference efficiency on edge devices. Specifically, they significantly increase computation workloads, resulting in longer inference latency. For example, to augment the dataset Cora [1], only 2% of the original edges are eliminated, while potential edges equivalent to 57% of the original edge count are added in [23]. Moreover, current methods only consider introducing potential edges with high importance, ignoring the irregular memory usage demands that follow. Despite the newly added edges, the connections still remain highly sparse, making it hard for edge devices to achieve efficient inference due to the few data reuse opportunities.

### C. Motivation

Although HyperGNNs outperform GNNs, their more complex computation patterns present challenges for efficient edge inference, resulting in increased inference latency. As mentioned in Section I, the long-latency issue originates from the two-stage aggregation phase, which induces irregular data usage demands and consequently results in numerous off-chip memory accesses. In essence, this issue arises from the highly sparse connections in hypergraph structures. Given that the underlying cause is inherent to hypergraphs, existing hardware

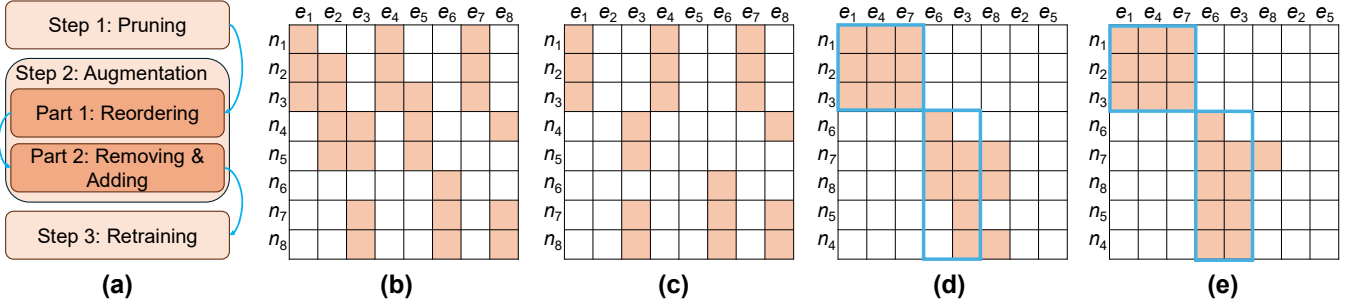


Fig. 2. Grin enables efficient inference on edge devices while preserving model performance: (a) Framework diagram; (b)  $\mathbf{H}$  of an original hypergraph; (c)  $\mathbf{H}$  after Step 1; (d)  $\mathbf{H}$  after Part 1 of Step 2, where blue rectangles refer to identified dense regions with enhanced data reuse; (e)  $\mathbf{H}$  after Part 2 of Step 2.

accelerator designs for HyperGNNs [13], [14] can only achieve limited performance gains by exploiting scarce data reuse opportunities, without addressing the fundamental limitation.

To address the underlying cause, the paper proposes a general HyperGNN training framework that restructures hypergraphs to enable efficient edge inference. Inspired by traditional data augmentation methods, this approach uniquely reconstructs hypergraphs to account for both model performance and inference efficiency jointly. This dual focus makes it particularly well-suited for practical implementation in real-world scenarios.

### III. GRIN

#### A. Overview

Since each connection (i.e., a non-zero element in  $H$ ) within hypergraphs represents a data usage demand in the aggregation phase, these highly sparse connections lead to irregular data usage demands together. To address the fundamental cause inherent in hypergraphs and to enable efficient inference on edge devices while enhancing model performance, Grin follows two essential requirements. For model performance, connections with high importance that contribute to data diversity are either preserved if they already exist or introduced if they are potentially beneficial, while low-importance ones are removed. For efficient inference on edge devices, Grin applies two key principles that guide the restructuring of hypergraphs to enhance their compatibility with hardware platforms.

- **Principle 1: Reduced memory usage demands compared to the original.** This principle ensures that the restructured hypergraphs do not generate more data usage demands than the original. Specifically, the number of newly added connections must be smaller than the number of removed ones. Although data usage demands still remain irregular, the overall number of off-chip memory accesses is reduced. This reduction helps to mitigate the long-latency issue in inference to some extent.
- **Principle 2: Enhanced data reuse compared to the original.** The principle focuses on improving data reuse of remaining data usage demands, aiming to minimize off-chip memory accesses and thus achieve efficient inference on edge devices. To accomplish this, connections that facilitate data reuse are either preserved or introduced, while those that hinder data reuse are eliminated.

As illustrated in Fig. 2(a), Grin comprises three sequential steps. Specifically, the first two steps focus on restructuring

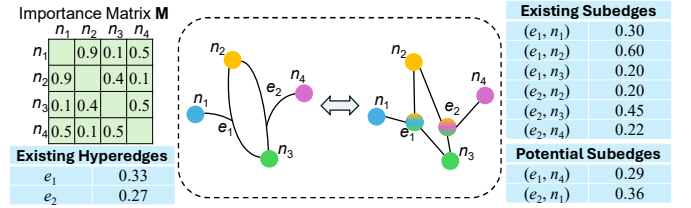


Fig. 3. Example of hyperedge importance and subedge importance evaluation.

hypergraphs to satisfy the previously defined principles, thereby addressing the underlying cause inherent to hypergraphs. To fulfill **Principle 1, Step 1 (Hypergraph Pruning)** removes unnecessary hyperedges to reduce computation workloads and performs initial training of the HyperGNN model parameters. **Step 2 (Hypergraph Augmentation)** augments the hypergraph structures to meet the requirements of **Principle 2**, and can be further divided into two parts. In Part 1, nodes and hyperedges are reordered to form regions with high connection density, which preliminarily improves data reuse. In Part 2, data reuse and model performance are further improved by introducing potential high-importance connections that are not originally present in the identified dense regions, and by eliminating low-importance connections from other sparse regions. Finally, **Step 3 (HyperGNN Retraining)** fine-tunes the model parameters trained in Step 1 based on the updated hypergraph structures.

The remainder of the section is organized as follows. To provide a clearer understanding of Grin, it begins with an introduction to a hypergraph analysis method derived from GAE-based graph augmentation methods. After establishing this foundation, each step of Grin is presented in detail.

#### B. Hypergraph Analysis Method

A notable feature of GAE-based graph augmentation methods is their use of an importance matrix  $\mathbf{M}$ , which assesses the relevance of both existing and potential pairwise connections for enhancing model performance. However, this matrix cannot be directly applied to analyze hypergraphs, as it is designed to evaluate pairwise node relationships that are absent in hypergraph structures. To bridge the gap, inspired by the observation that current hypergraphs are primarily constructed from graphs utilizing specific strategies [20], a GAE-based hypergraph analysis method is proposed. This method forms the foundation of Grin by enabling the evaluation of both hyperedge importance and subedge importance in hypergraphs.

**Hyperedge Importance Evaluation.** The evaluation aims to determine the importance of each hyperedge for its contribution

to model performance. To achieve this, the nodes connected by the target hyperedge are collected. Then, the geometric mean of the importance values for all possible node pairs within this set is utilized to quantify the hyperedge’s importance. For instance, to evaluate  $e_1$  in the hypergraph in Fig. 3, the geometric mean of  $\mathbf{M}[n_1, n_2]$ ,  $\mathbf{M}[n_1, n_3]$ , and  $\mathbf{M}[n_2, n_3]$  is used.

**Subedge Importance Evaluation.** The evaluation focuses on assessing the importance of each component within a hyperedge. In this context, each hyperedge is treated as a special node, and its connection to one of its associated nodes is referred to as a subedge. For example, the initial hyperedge  $e_1$  in the hypergraph depicted in Fig. 3 consists of three subedges:  $(e_1, n_1)$ ,  $(e_1, n_2)$ , and  $(e_1, n_3)$ . Since each subedge is part of its corresponding hyperedge, its importance is evaluated based on the structure of that hyperedge. Specifically, the importance of a subedge is quantified by computing the geometric mean of the importance values between the node in the target subedge and all other nodes connected to the same hyperedge. For instance, to evaluate  $(e_1, n_1)$  in the hypergraph of Fig. 3, the geometric mean of  $\mathbf{M}[n_1, n_2]$ ,  $\mathbf{M}[n_1, n_3]$  is calculated. Similarly, potential subedges that are not present in the original hypergraph can be measured using the same approach.

### C. Grin Framework

**Step 1: Hypergraph Pruning.** Step 1 aims to prune unnecessary hyperedges and mitigate overall data usage demands, thereby satisfying Principle 1. This step involves a training process that simultaneously initializes model parameters and identifies hyperedges for elimination. Initially, each hyperedge is assigned an equal weight. The magnitude of this weight serves as an indicator of the hyperedge’s impact on model performance. During training, both the model parameters and hyperedge weights are updated jointly. Upon completion, hyperedges with the lowest weight magnitudes are removed according to the target prune ratio  $tp_r$ .

To demonstrate the changes resulting from Step 1, Fig. 2(b) and Fig. 2(c) present an example. Fig. 2(b) displays the incidence matrix  $\mathbf{H}$  of an initial hypergraph containing 8 nodes and 8 hyperedges. After training, the weight magnitudes of  $e_2$  and  $e_5$  are the lowest among all hyperedges, indicating that their contributions are less significant compared to the others. Therefore, both hyperedges are removed to reduce data usage demands as depicted in Fig. 2(c).

**Part 1 of Step 2: Hypergraph Reordering.** Part 1 focuses on altering the execution order of hyperedges and nodes to improve data reuse, thereby supporting the objective of Principle 2. In real-world hypergraphs, the indexing of nodes and hyperedges is typically random, which limits opportunities for data reuse when inference strictly follows these indices [28]. To address the problem, it is essential to reorder hypergraphs based on the connection relationships between nodes and hyperedges. Such ordering naturally promotes data reuse, as shared nodes or hyperedges create overlapping data usage demands. After reordering, regions with significantly higher connection density emerge in  $\mathbf{H}$ . These regions are referred to as dense regions (whose specific identification method is discussed in Part 2) and indicate improved data reuse. Since most subedges in the

---

### Algorithm 1 Hypergraph Reordering Algorithm

---

**Require:**

node list:  $N$ ; hyperedge list:  $E$ ; threshold:  $thre$

**Ensure:**

reordered node list:  $RN$

reordered hyperedge list:  $RE$

```

1:  $RN = []$ ;  $RE = []$ 
2: set  $Dict$  to record hyperedge importance values
3: while  $N \neq []$  do
4:    $counter = 0$ ;  $Dict = \{\}$ 
5:   randomly select a node  $n$  from  $N$ 
6:   move  $n$  to  $RN$  from  $N$ 
7:    $HyG = \text{getHyperedgeGroup}(n)$ 
8:    $\text{update}(Dict, HyG, n)$ 
9:   while  $counter \leq thre$  and  $Dict \neq \{\}$  do
10:     $e = \text{getMaxImportanceHyperedgeId}(Dict)$ 
11:    remove  $e$  and its corresponding importance from  $Dict$ 
12:    move  $e$  to  $RE$  from  $E$ 
13:     $counter += \text{getSubedgeNumber}(e)$ 
14:     $NG = \text{getNodeGroup}(e)$ 
15:    move nodes in  $NG$  from  $N$  to  $RN$ 
16:     $HyG = \text{getHyperedgeGroup}(NG)$ 
17:     $\text{update}(Dict, HyG, NG)$ 
18:   end while
19: end while
20: move the remaining hyperedges in  $E$  to  $RE$ 
21: return  $RN, RE$ 

```

---

remaining sparse regions will be removed in Part 2, an advanced reordering algorithm is proposed to concentrate as many high-importance subedges as possible in the resulting dense regions.

Algorithm 1 presents the proposed algorithm. The algorithm scans the input hypergraph based on connection relationships and prioritizes the reordering of high-importance hyperedges to concentrate high-importance subedges. Specifically, Algorithm 1 begins by randomly selecting a node (Line 5). This node is then removed from  $N$  and added to the reordered node list (Line 6). Subsequently, the hyperedges connected to this node are retrieved (Line 7) to update the importance dictionary  $Dict$  with the current hyperedges (Line 8). To reduce execution time, the importance of a hyperedge is simply evaluated using the minimum importance value among the scanned subedges. Once completed, the hyperedge with the highest importance is added to the reordered hyperedge list (Line 10-12). After a hyperedge is added, its connected nodes are obtained (Line 14) and directly appended to the reordered node list (Line 15). These newly added nodes are used to retrieve their associated hyperedges, which are then used to update  $Dict$  (Lines 16–17) for selecting the next hyperedge to include in  $RE$ .

Within Algorithm 1,  $counter$  tracks the number of subedges included in the reordered hypergraph (Line 13). When  $counter$  exceeds a predefined threshold  $thre$ , a new reordering process is initiated based on the remaining unprocessed hypergraph (Line 9). This step is essential because a larger value of  $thre$  tends to reduce the connection density in the resulting dense regions, which in turn limits data reuse opportunities. Moreover, a new reordering process also starts when no additional

unscanned hyperedges are available, to prevent infinite loops (Line 9). This situation may arise when separated regions appear, caused by elements being removed from  $N$  or  $E$ . After reordering all nodes, remaining hyperedges in  $E$  are directly transferred to  $RE$  to complete Algorithm 1 (Line 20).

Fig. 2(c) gives an example to illustrate the changes introduced in Part 1, with  $thre$  set to 6 and  $n_1$  selected as the initial node. The hyperedges connected to  $n_1$  are  $e_1$ ,  $e_4$ , and  $e_7$ . Assuming that  $e_1$  features the highest importance, its associated nodes,  $n_2$  and  $n_3$ , are subsequently reordered. Then the hyperedges connected to these two nodes,  $e_4$  and  $e_7$ , are used to update  $Dict$ . Once  $e_4$  and  $e_7$  are added to  $RE$ , the total number of subedges exceeds  $thre$ , prompting the initiation of a new reordering process. For the remaining hypergraph,  $n_6$  is selected first. Since  $e_6$  is the only hyperedge connected to  $n_6$ , it is appended to  $RE$ , and its associated nodes  $n_7$  and  $n_8$  are added to  $RN$ . Assuming that  $e_3$  features a higher importance than  $e_8$ ,  $e_3$  is then added to  $RE$ . At this point, the reordering process concludes because the subedge number becomes 7, which exceeds  $thre$ . Finally, since all the nodes have been reordered, the remaining  $e_8$  is directly added to  $RE$ . Compared to Fig. 2(c), the subedge distribution shown in Fig. 2(d) can be accessed more successively during the aggregation phase, indicating increased data reuse and regular data usage demands.

**Part 2 of Step 2: Subedge Removing & Adding.** Following Part 1, although data reuse is improved partially, a substantial amount of irregular data usage demands still remains. Hence, Part 2 aims to further enhance data reuse to fulfill Principle 2. This is achieved by introducing new high-importance subedges in dense regions generated during reordering and eliminating low-importance subedges from the remaining sparse regions.

Within Part 2, the reordered hypergraph produced in Part 1 is first partitioned into dense and sparse regions. Specifically, a dense region is formed by the nodes and hyperedges that are reordered during a reordering process to achieve high connection density. Since Algorithm 1 performs multiple reordering processes, multiple dense regions are formed. For instance, as illustrated by the blue rectangles in Fig. 2(d), two dense regions are generated due to the execution of two reordering processes:  $\langle n_1, n_2, n_3, e_1, e_4, e_7 \rangle$  and  $\langle n_6, n_7, n_8, n_5, n_4, e_6, e_3 \rangle$ . All areas outside these regions are classified as sparse regions.

To further enhance data reuse, Part 2 restricts the addition of new subedges to dense regions and the removal of subedges to sparse regions. Note that although subedges with high importance are mainly concentrated in dense regions from Part 1, a small number of them still exist in sparse regions. To improve model performance, the subedges with the highest importance values in sparse regions, corresponding to the top target keep ratio  $tkr$ , are retained. The remaining subedges in these regions are removed to enhance data reuse. When introducing new subedges in dense regions, the number added is kept equal to the number removed from sparse regions to avoid increased computation workloads. Let  $n_{ss}$  denote the number of subedges in sparse regions. The potential subedges in dense regions corresponding to the top  $n_{ss} \times (1 - tkr)$  importance values are then selected for addition. It is important to emphasize that

TABLE I  
DATASET INFORMATION

	Cora	Citeseer	Pubmed	DBLP	ACM
# Nodes	2,708	3,327	19,717	4,057	3,025
# Hyperedges	2,590	2,996	19,711	3,625	2,231
# Features	1,433	3,703	500	334	1,870
# Classes	7	6	3	4	3
# Training nodes	140	120	60	405	302
# Validation nodes	500	500	500	406	303
# Test nodes	1,000	1,000	1,000	3,246	2,420

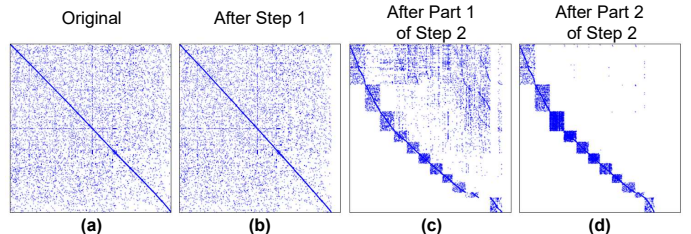


Fig. 4. Illustration of Grin on Cora: (a)  $\mathbf{H}$  of Cora; (b)  $\mathbf{H}$  after Step 1; (c)  $\mathbf{H}$  after Part 1 of Step 2; (d)  $\mathbf{H}$  after Part 2 of Step 2.

dense regions occupy only a small fraction of the overall space in  $\mathbf{H}$ . Hence, the time required to collect importance values for potential subedges is significantly reduced.

Fig. 2(d) and Fig. 2(e) demonstrate an example of how Part 2 works. Initially, there are three subedges in sparse regions. To keep the high-importance subedge among them, the subedge  $(e_8, n_7)$  is retained. Therefore, the other two subedges are removed to enhance data reuse. In dense regions, three potential subedges exist. Since the number of added subedges should be the same as the number of removed subedges, two subedges with higher importance values,  $(e_6, n_5)$  and  $(e_6, n_4)$ , are added to improve model performance and further enhance data reuse.

**Step 3: HyperGNN Retraining.** Since Step 2 optimizes hypergraph structures independently of the model parameters trained in Step 1, Step 3 is dedicated to retraining those parameters based on the updated hypergraphs. This step fine-tunes the parameters to ensure they are properly aligned with the structural changes introduced in Step 2.

## IV. EVALUATION

### A. Setup

**Datasets.** There are 5 academic citation datasets used for node classification tasks: Cora, Citeseer [1], Pubmed [1], DBLP [29], and ACM [29]. To build hypergraphs, the 1-hop neighbor-based hyperedge construction method [20] is utilized. The information about the above datasets is summarized in Table I.

**Models.** Three widely used HyperGNN models, HGNN [18], UniGIN [19], and HGNN+ [20], are employed to evaluate Grin for model performance and inference efficiency. Each has a two-layer structure, with a hidden dimension of 16. When training models, a semi-supervised method is employed, following the approach used in most related works [18]–[20].

**Hardware Platforms.** An NVIDIA A40 GPU is utilized to train HyperGNN models with Grin. To evaluate inference efficiency, since all of the current HyperGNN accelerators follow the same principle of leveraging available data reuse opportunities, a cycle-accurate simulator of the representative SOTA accelerator MeHyper [14] is developed. The same configurations in [14] are adopted for the simulator.

TABLE II  
MODEL ACCURACY RESULTS OF GRIN (%)

Model	Method	Cora	Citeseer	Pubmed	DBLP	ACM
HGNN	Baseline	81.33	71.46	78.46	77.28	86.40
	Grin	<b>81.93</b>	<b>71.49</b>	<b>79.01</b>	<b>78.42</b>	<b>88.03</b>
UniGIN	Baseline	77.63	65.49	77.07	75.74	81.64
	Grin	<b>79.23</b>	<b>67.35</b>	<b>77.33</b>	<b>78.04</b>	<b>83.27</b>
HGNN+	Baseline	81.10	69.79	77.15	<b>76.74</b>	85.65
	Grin	<b>81.57</b>	<b>70.30</b>	<b>79.01</b>	76.50	<b>86.01</b>

TABLE III  
OPTIMAL CONFIGURATIONS OF GRIN

Model	Parameter	Cora	Citeseer	Pubmed	DBLP	ACM
HGNN	<i>tpr</i>	0.09	0.05	0	0.08	0.30
	<i>tkr</i>	0.02	0.005	0.005	0.01	0.04
UniGIN	<i>tpr</i>	0.07	0.07	0	0.30	0.28
	<i>tkr</i>	0.10	0	0.10	0	0.01
HGNN+	<i>tpr</i>	0	0.02	0	0.04	0.16
	<i>tkr</i>	0.10	0	0.10	0.10	0.04

## B. Visualization

Fig. 4 illustrates the transformation of  $\mathbf{H}$  in the Cora dataset from Step 1 to Step 2. *tpr*, *tkr*, and *thre*, are set to 0.1, 0.01, and 1,000, respectively. After processing with Grin, the subedge distribution becomes significantly more concentrated compared to the original, with only a small number of subedges in sparse regions retained to preserve high-importance ones.

## C. Model Performance

Model performance is evaluated using accuracy on the datasets in Table I, with original hypergraphs as the baseline. To configure Grin, *thre* is fixed at 1,000, while *tpr* and *tkr* are selected from the ranges 0 to 0.3 and 0 to 0.1, respectively.

Table II presents the accuracy results averaged over 10 independent runs. Grin is configured using the optimal settings in Table III. For each dataset and model combination case, the higher accuracy value between the baseline and Grin is highlighted in bold. From Table II, Grin achieves an average accuracy increase of 0.97%, indicating that it enhances model performance by introducing high-importance subedges and eliminating low-importance ones. The restructuring increases data diversity and contributes to robustness. Moreover, Grin outperforms the baseline in most cases, with the most notable improvement being a 2.30% increase for DBLP under UniGIN. The only decrease occurs in DBLP under HGNN+, where the accuracy drops slightly by 0.24% compared to the baseline.

## D. Inference Speedup

To evaluate the inference efficiency enhanced by Grin, both original hypergraphs and intermediate hypergraphs after Step 1 of Grin are selected for comparison. The results are illustrated in Fig. 5, with each case configured using the optimal parameters specified in Table III. In Fig. 5, the inference latency when deploying original hypergraphs on MeHyper serves as the baseline, represented by the red horizontal dashed line. Specifically, deploying Grin-optimized hypergraphs on MeHyper achieves an average inference speedup of 1.41 $\times$  over the baseline, while reducing off-memory access by 27.60% on average. Given that MeHyper itself achieves a 10.51 $\times$  average speedup over the SOTA GPU-based system HyperGef [30], the integration of Grin-optimized hypergraphs on MeHyper yields a cumulative inference speedup of 14.82 $\times$  over HyperGef.

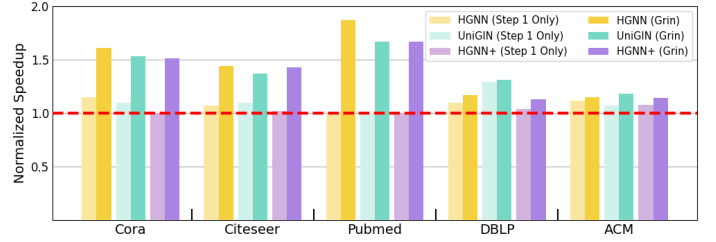


Fig. 5. Inference speedup results of Grin.

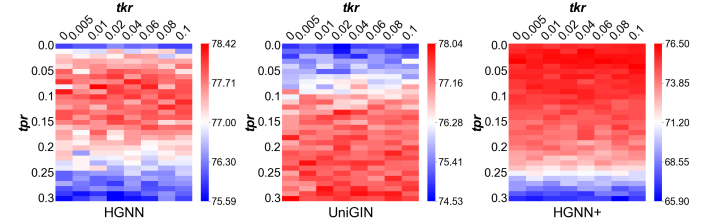


Fig. 6. Model accuracy results under various configurations for DBLP.

These results indicate that Grin efficiently improves inference efficiency. Moreover, the performance gain from executing Step 1 alone is limited to a mere 1.08 $\times$ . This modest improvement is due to the constraint that Step 1 cannot remove numerous hyperedges to preserve model accuracy. Meanwhile, when Step 2 is incorporated, Grin exhibits a speedup of 1.31 $\times$  compared to Step 1 alone. This demonstrates that the proposed augmentation method effectively enhances data reuse for efficient inference.

## E. Ablation Study

To assess the impact of different configurations of *tpr* and *tkr* on model accuracy, DBLP is used for evaluation. Accuracy results under three models are illustrated in Fig. 6, where *tpr* is uniformly distributed in the range of 0-0.3 with an interval of 0.01. In contrast, smaller values of *tkr* are preferred to reduce the number of subedges in sparse regions for promoting data reuse. This preference results in an unevenly distributed range of *tkr*, with denser sampling concentrated near 0.

In Fig. 6, the horizontal changes at a fixed value of *tpr* are relatively modest, indicating that the newly introduced subedges in Grin possess sufficient importance to offset the impact of removed original subedges. In contrast, the vertical changes as *tpr* varies in Fig. 6 are more pronounced. Accuracy exhibits distinct behaviors under three models. For HGNN, accuracy initially improves with increasing *tpr* but declines beyond a certain threshold. UniGIN consistently increases in accuracy as *tpr* rises, while HGNN+ experiences a steady decline. These ablation results underscore that different models respond differently to changes in *tpr*. Therefore, careful selection of *tpr* is essential for achieving optimal model accuracy with Grin.

## V. CONCLUSION

This paper proposes Grin, a general HyperGNN training framework that restructures hypergraphs for efficient edge inference regardless of hardware architectures while improving model performance. Experiments highlight the benefits of Grin.

## VI. ACKNOWLEDGEMENT

This work was partially supported by Hong Kong Research Grants Council General Research Fund (Grant No. 16213422) and Collaborative Research Fund (Grant No. C5032-23G).

## REFERENCES

- [1] T. N. Kipf and M. Welling, "Semi-supervised classification with graph convolutional networks," *arXiv preprint arXiv:1609.02907*, 2016.
- [2] W. Hamilton, Z. Ying, and J. Leskovec, "Inductive representation learning on large graphs," *Advances in neural information processing systems*, vol. 30, 2017.
- [3] P. Veličković, G. Cucurull, A. Casanova, A. Romero, P. Lio, and Y. Bengio, "Graph attention networks," *arXiv preprint arXiv:1710.10903*, 2017.
- [4] Z. Wu, S. Pan, F. Chen, G. Long, C. Zhang, and P. S. Yu, "A comprehensive survey on graph neural networks," *IEEE transactions on neural networks and learning systems*, vol. 32, no. 1, pp. 4–24, 2020.
- [5] J. Jiang, Y. Wei, Y. Feng, J. Cao, and Y. Gao, "Dynamic hypergraph neural networks," in *IJCAI*, 2019, pp. 2635–2641.
- [6] A. Antelmi, G. Cordasco, M. Polato, V. Scarano, C. Spagnuolo, and D. Yang, "A survey on hypergraph representation learning," *ACM Computing Surveys*, vol. 56, no. 1, pp. 1–38, 2023.
- [7] S. Kim, S. Y. Lee, Y. Gao, A. Antelmi, M. Polato, and K. Shin, "A survey on hypergraph neural networks: An in-depth and step-by-step guide," in *Proceedings of the 30th ACM SIGKDD Conference on Knowledge Discovery and Data Mining*, 2024, pp. 6534–6544.
- [8] X. Hao, J. Li, Y. Guo, T. Jiang, and M. Yu, "Hypergraph neural network for skeleton-based action recognition," *IEEE Transactions on Image Processing*, vol. 30, pp. 2263–2275, 2021.
- [9] J. Wang, K. Ding, L. Hong, H. Liu, and J. Caverlee, "Next-item recommendation with sequential hypergraphs," in *Proceedings of the 43rd international ACM SIGIR conference on research and development in information retrieval*, 2020, pp. 1101–1110.
- [10] B. Khan, J. Wu, J. Yang, and X. Ma, "Heterogeneous hypergraph neural network for social recommendation using attention network," *ACM Transactions on Recommender Systems*, vol. 3, no. 3, pp. 1–22, 2025.
- [11] S. Feng, E. Heath, B. Jefferson, C. Joslyn, H. Kvinge, H. D. Mitchell, B. Praggastis, A. J. Eisfeld, A. C. Sims, L. B. Thackray *et al.*, "Hypergraph models of biological networks to identify genes critical to pathogenic viral response," *BMC bioinformatics*, vol. 22, no. 1, p. 287, 2021.
- [12] X. Hao, J. Li, M. Ma, J. Qin, D. Zhang, F. Liu, A. D. N. Initiative *et al.*, "Hypergraph convolutional network for longitudinal data analysis in alzheimer's disease," *Computers in Biology and Medicine*, vol. 168, p. 107765, 2024.
- [13] H. Yu, Y. Zhang, L. He, Y. Zhao, X. Li, R. Xin, J. Zhao, X. Liao, H. Liu, B. He *et al.*, "Rahp: A redundancy-aware accelerator for high-performance hypergraph neural network," in *2024 57th IEEE/ACM International Symposium on Microarchitecture (MICRO)*. IEEE, 2024, pp. 1264–1277.
- [14] W. Zhao, P. Yao, D. Chen, L. Zheng, X. Liao, Q. Wang, S. Ma, Y. Li, H. Liu, W. Xiao *et al.*, "Mehyper: Accelerating hypergraph neural networks by exploring implicit dataflows," in *2025 IEEE International Symposium on High Performance Computer Architecture (HPCA)*. IEEE, 2025, pp. 920–933.
- [15] T. Zhao, W. Jin, Y. Liu, Y. Wang, G. Liu, S. Günemann, N. Shah, and M. Jiang, "Graph data augmentation for graph machine learning: A survey," *arXiv preprint arXiv:2202.08871*, 2022.
- [16] Z. Wang, P. Wang, K. Liu, P. Wang, Y. Fu, C.-T. Lu, C. C. Aggarwal, J. Pei, and Y. Zhou, "A comprehensive survey on data augmentation," *arXiv preprint arXiv:2405.09591*, 2024.
- [17] J. Zhou, C. Xie, S. Gong, Z. Wen, X. Zhao, Q. Xuan, and X. Yang, "Data augmentation on graphs: A technical survey," *ACM Computing Surveys*, vol. 57, no. 11, pp. 1–34, 2025.
- [18] Y. Feng, H. You, Z. Zhang, R. Ji, and Y. Gao, "Hypergraph neural networks," in *Proceedings of the AAAI conference on artificial intelligence*, vol. 33, no. 01, 2019, pp. 3558–3565.
- [19] J. Huang and J. Yang, "Unignn: a unified framework for graph and hypergraph neural networks," in *Proceedings of the Thirtieth International Joint Conference on Artificial Intelligence*. International Joint Conferences on Artificial Intelligence Organization, 2021.
- [20] Y. Gao, Y. Feng, S. Ji, and R. Ji, "Hggn+: General hypergraph neural networks," *IEEE Transactions on Pattern Analysis and Machine Intelligence*, vol. 45, no. 3, pp. 3181–3199, 2022.
- [21] H. Park, S. Lee, S. Kim, J. Park, J. Jeong, K.-M. Kim, J.-W. Ha, and H. J. Kim, "Metropolis-hastings data augmentation for graph neural networks," *Advances in Neural Information Processing Systems*, vol. 34, pp. 19010–19020, 2021.
- [22] I. Spinelli, S. Scardapane, A. Hussain, and A. Uncini, "Fairdrop: Biased edge dropout for enhancing fairness in graph representation learning," *IEEE Transactions on Artificial Intelligence*, vol. 3, no. 3, pp. 344–354, 2021.
- [23] T. Zhao, Y. Liu, L. Neves, O. Woodford, M. Jiang, and N. Shah, "Data augmentation for graph neural networks," in *Proceedings of the AAAI conference on artificial intelligence*, vol. 35, no. 12, 2021, pp. 11015–11023.
- [24] T. Wei, Y. You, T. Chen, Y. Shen, J. He, and Z. Wang, "Augmentations in hypergraph contrastive learning: Fabricated and generative," *Advances in neural information processing systems*, vol. 35, pp. 1909–1922, 2022.
- [25] D. Cai, M. Song, C. Sun, B. Zhang, S. Hong, and H. Li, "Hypergraph structure learning for hypergraph neural networks," in *IJCAI*, 2022, pp. 1923–1929.
- [26] Y. Song, Y. Gu, T. Li, J. Qi, Z. Liu, C. S. Jensen, and G. Yu, "Chggn: A semi-supervised contrastive hypergraph learning network," *IEEE Transactions on Knowledge and Data Engineering*, vol. 36, no. 9, pp. 4515–4530, 2024.
- [27] T. N. Kipf and M. Welling, "Variational graph auto-encoders," *arXiv preprint arXiv:1611.07308*, 2016.
- [28] H. You, T. Geng, Y. Zhang, A. Li, and Y. Lin, "Gcod: Graph convolutional network acceleration via dedicated algorithm and accelerator co-design," in *2022 IEEE International Symposium on High-Performance Computer Architecture (HPCA)*. IEEE, 2022, pp. 460–474.
- [29] Y. Liu, W. Tu, S. Zhou, X. Liu, L. Song, X. Yang, and E. Zhu, "Deep graph clustering via dual correlation reduction," in *Proceedings of the AAAI Conference on Artificial Intelligence*, vol. 36, no. 7, 2022, pp. 7603–7611.
- [30] Z. Yu, G. Dai, S. Yang, G. Zhang, H. Zhang, F. Zhu, J. Yang, J. Zhao, and Y. Wang, "Hypergef: A framework enabling efficient fusion for hypergraph neural network on gpus," *Proceedings of Machine Learning and Systems*, vol. 5, pp. 387–399, 2023.