Data Pruning-enabled High Performance and Reliable Graph Neural Network Training on ReRAM-based Processing-in-Memory Accelerators

CHUKWUFUMNANYA OGBOGU, Washington State University, Pullman, United States
BIRESH JOARDAR, University of Houston System, Houston, United States
KRISHNENDU CHAKRABARTY, Arizona State University, Tempe, United States
JANA DOPPA, Washington State University, Pullman, United States
PARTHA PRATIM PANDE, Washington State University, Pullman, United States

Graph Neural Networks (GNNs) have achieved remarkable accuracy in cognitive tasks such as predictive analytics on graph-structured data. Hence, they have become very popular in diverse real-world applications. However, GNN training with large real-world graph datasets in edge-computing scenarios is both memory- and compute-intensive. Traditional computing platforms such as CPUs and GPUs do not provide the energy efficiency and low latency required in edge intelligence applications due to their limited memory bandwidth. Resistive random-access memory (ReRAM)-based processing-in-memory (PIM) architectures have been proposed as suitable candidates for accelerating AI applications at the edge, including GNN training. However, ReRAM-based PIM architectures suffer from low reliability due to their limited endurance, and low performance when they are used for GNN training in real-world scenarios with large graphs. In this work, we propose a learning-for-data-pruning framework, which leverages a trained Binary Graph Classifier (BGC) to reduce the size of the input data graph by pruning subgraphs early in the training process to accelerate the GNN training process on ReRAM-based architectures. The proposed light-weight BGC model reduces the amount of redundant information in input graph(s) to speed up the overall training process, improves the reliability of the ReRAM-based PIM accelerator, and reduces the overall training cost. This enables fast, energy-efficient, and reliable GNN training on ReRAM-based architectures. Our experimental results demonstrate that using this learning for data pruning framework, we can accelerate GNN training and improve the reliability of ReRAM-based PIM architectures by up to 1.6×, and reduce the overall training cost by 100× compared to state-of-the-art data pruning techniques.

CCS Concepts: • **Hardware → Emerging technologies**; **Analysis and design of emerging devices and systems**; **Emerging architectures**;

Additional Key Words and Phrases: Performance, reliability, non-volatile memory, endurance

This work was supported, in part by the US National Science Foundation (NSF) under grants CNS-1955353, CNS-1955196 and CNS-2308530.

Authors' Contact Information: Chukwufumnanya Ogbogu, Washington State University, Pullman, Washington, United States; e-mail: c.ogbogu@wsu.edu; Biresh Joardar, University of Houston System, Houston, Texas, United States; e-mail: bjoardar@Central.UH.EDU; Krishnendu Chakrabarty, Arizona State University, Tempe, Arizona, United States; e-mail: krishnendu.chakrabarty@asu.edu; Jana Doppa, Washington State University, Pullman, Washington, United States; e-mail: jana.doppa@wsu.edu; Partha Pratim Pande, Washington State University, Pullman, Washington, United States; e-mail: pande@wsu.edu.

ACM 1084-4309/2024/08-ART72
https://doi.org/10.1145/3656171

ACM Reference Format:
Chukwufumnanya Ogbogu, Biresh Joardar, Krishnendu Chakrabarty, Jana Doppa, and Partha Pratim Pande. 2024. Data Pruning-enabled High Performance and Reliable Graph Neural Network Training on ReRAM-based Processing-in-Memory Accelerators. *ACM Trans. Des. Autom. Electron. Syst.* 29, 5, Article 72 (August 2024), 29 pages. https://doi.org/10.1145/3656171

1 INTRODUCTION

Graph Neural Networks (GNNs) have achieved notable success in a large range of applications, and they have become mainstream for performing cognitive tasks (e.g., node classification, link/edge prediction, and graph classification) on graph structured data, which are ubiquitous in diverse application domains. Examples of such domains and their applications include chemistry (e.g., designing compounds), medicine (e.g., experimental drug-discovery), transportation (e.g., finding efficient routes), and so on [1, 2]. However, the exponential growth in computation cost as graph datasets grow in size (up to billions of nodes and edges [1]) and GNN model structures become more complex, present a significant challenge. The massive computational and storage requirements for processing real-world graph data limit the deployment of GNN models on resource-constrained and time-sensitive computing platforms such as IoT devices in edge AI applications [1, 3, 4]. Moreover, traditional architectures (such as CPUs, GPUs and TPUs) do not provide the compute efficiency (FLOPs/W) and low latency required for edge AI applications such as GNN training and inference due to their limited memory bandwidth and high-power consumption [5].

To address this challenge, domain-specific architectures such as **processing in memory (PIM)**-based computing platforms have emerged as suitable candidates for edge AI applications [6–8]. **Resistive random-access memory (ReRAM)**-based crossbar arrays enable energy-efficient and high-performance **Matrix Vector Multiplication (MVM)** operations [9]. MVM is the core computational kernel of GNN computation. Hence, ReRAM-based PIM architectures are suitable for GNN computation. However, the high computational requirements imposed by large-scale graph datasets still make executing GNN models on PIM-based architectures very challenging. In addition, PIM architectures based on ReRAMs suffer from low write endurance, which significantly limits the number of times they can be re-programmed before they fail due to faults [10, 11]. Therefore, it is important to explore approaches that reduce the number of ReRAM write operations arising from the high computation cost associated with GNN training tasks for large graph datasets. This will enable more GNN training instances to be performed on PIM architectures.

To reduce the high computation cost of GNN training, approaches such as **Unified Graph Sparsification (UGS)** [12] and **Graph early-bird tickets (GEBT)** [13] for weight- and graph pruning have been developed. Both UGS and GEBT are based on the **Lottery Ticket Hypothesis (LTH)** [13, 14]. Specifically, these methods attempt to generalize the LTH (originally proposed for CNNs) to GNNs, for pruning both the graph dataset and model weights with little accuracy loss [12, 13]. However, LTP-based methods don't take the ReRAM crossbar structure into consideration. As a result, there is no significant area or power savings. Recently, an LTH-inspired **crossbar-aware model pruning (CAP)** method for GNN training on ReRAM-based architectures was proposed in [15]. This method enhances existing CAP methods by taking the overall crossbar structure into consideration and implements model pruning in an iterative manner to ensure that the pruned GNN models can be trained from scratch. However, LTH-based approaches for GNNs usually adopt an iterative cycle (train-prune-retrain) that require multiple training rounds to find a sparse GNN model and graph dataset pair for the purpose of reducing both the training and inference cost [3]. As we show later, this approach is very expensive and increases the overall training cost by over 20×. In this work, we specifically define cost in terms of the training time required to find the

maximally pruned graph dataset that can be trained with a given GNN model without any accuracy loss. In order to reduce the cost of data graph pruning with negligible loss in accuracy, there is a pressing need to explore low-cost data pruning approaches to enable resource efficient graph learning.

In this work, we propose a novel data graph pruning approach that enables the training of GNNs using a fraction of the original graph dataset, with negligible loss in accuracy. The proposed method leverages a low-cost graph classification technique to reliably predict input subgraphs from a monolithic graph that can be pruned for GNN training in resource-constrained scenarios without any accuracy loss in the predictive accuracy of the trained GNN model after data pruning. This low-cost graph classification approach leverages a small **Multi-layer Perceptron (MLP)** classifier to learn patterns over subgraph structures and predict if a given subgraph should be pruned or retained during GNN training. In short, this approach can be conceptualized as a *learning-for-data pruning* paradigm, where the graph classifier is pre-trained offline to learn how to prune an unseen graph dataset during the early part of online GNN training. For the sake of brevity, we refer to this MLP-based subgraph classifier for efficient GNN training as the **Binary Graph Classifier (BGC)** throughout the rest of this paper. Due to the simple MLP-based structure, the execution of the BGC algorithm introduces an insignificant timing penalty to the total GNN training time as we show in our experimental analysis. Moreover, it is implemented offline on the host device. Overall, our approach reduces the overall computational cost by scaling-down the size of the graph dataset, reduces the number of ReRAM cell write operations, and improves the performance of GNN training on PIM-based architectures without compromising the predictive test accuracy of the trained GNN model. Consequently, this enables reliable GNN training without the occurrence of untimely ReRAM device breakdown or malfunction due to cell-wear induced faults. For the scope of this work, we consider PIM architectures based on ReRAM devices as a specific embodiment, as they are known to suffer more from the low endurance problem compared to other **non-volatile memory (NVM)** devices [9]. However, the proposed BGC-enabled graph pruning technique is applicable to other NVM-based PIM based architectures, as well as conventional CMOS-based CPU, GPU and TPU-based computing platforms. However, unlike NVM-based PIM endurance is not a major problem in CMOS-based architectures.

To evaluate the effectiveness of our proposed BGC-enabled GNN training framework, we perform extensive experiments across a variety of GNN models and large graph datasets. We demonstrate that our proposed approach is GNN model- and graph dataset-agnostic and can be generalized to even unseen graph datasets and other types of GNN models. In addition, we also show that our approach achieves the same level of performance as LTH-based pruning approaches while significantly reducing the overall GNN training cost. In particular, BGC-enabled GNN training significantly reduces the number of write operations to the ReRAM crossbar during training, thereby increasing the number of GNN training tasks that can be reliably executed on ReRAM-based PIM architectures, while ensuring performance speedup. We note that the proposed method can also be combined with existing ReRAM endurance-aware techniques (e.g., gradient sparsification and low-rank-training) to offer even higher improvements in the reliability for GNN training. Overall, our contributions in this paper can be highlighted as follows:

- We propose a novel approach to create **Binary Graph Classifier (BGC)** models to prune subgraphs during the GNN training process to reduce the overall training cost, improve performance and reliability without compromising the GNN test accuracy on ReRAM-based PIM architectures.
- We establish a trade-off between training cost, reliability, predictive accuracy, and performance for BGC-enabled subgraph pruning for GNN training on ReRAM-based PIM accelerators.

- We show that the proposed BGC-enabled subgraph pruning achieves up to 1.7× and 1.5× improvement in performance speedup and reliability respectively, and a reduction of up to 120× in training cost compared other existing LTH-based pruning methodologies on ReRAM-based GNN accelerators.

To the best of our knowledge, this is the first work that leverages the idea of subgraph pruning to enable resource-efficient GNN training on PIM-based architectures. The rest of this paper is organized as follows. Section 2 discusses relevant prior work. Section 3 elaborates the methodology for BGC-enabled subgraph pruning. Section 4 presents the results of the comprehensive experiments, and Section 5 concludes the paper by summarizing the key features of this work.

2 RELATED PRIOR WORK

In this section, we present some relevant prior work. GNNs leverage neighborhood aggregation to learn representations for nodes, edges, and entire graphs using various state of the art models such as **Graph Convolution Networks (GCNs)** [2], **Graph Attention Networks (GAT)** [16], and GraphSAGE [17]. In addition, these GNN models are often trained on a suite of very large graph datasets, each of which consists of millions/billions of nodes and edges with their associated features and have achieved remarkable accuracy. However, while the training of these GNN models with large graphs has been shown to be feasible on traditional CPU/GPU systems, these models cannot be easily adapted for training on resource-constrained computing platforms. Moreover, the computational overhead involved with GNN training on PIM-based accelerators is often not considered. Hence, it is important to understand the challenges associated with resource-constrained platforms and propose techniques to address them in order to enable resource-efficient GNN training.

In particular, ReRAM-based crossbars have been demonstrated to efficiently accelerate **Matrix-Vector-Multiplication (MVM)** operations, which are the predominant operations in GNN training and inferencing [7]. Recent work has proposed a 3D ReRAM-based PIM architecture to address the on-chip communication challenges of GNN training [18, 19]. However, both training and inferencing are affected by the non-ideal nature of ReRAM cells [9, 20]. ReRAM-based systems have low precision and limited write endurance that can affect the accuracy of the GNN model. Software-level optimization techniques such as dropout, quantization, and pruning can also be used to improve performance and energy-efficiency of GNN training on ReRAM-based architectures [6, 15]. However, the low write endurance problem of ReRAM devices remains a significant challenge and limits the feasibility of these proposed techniques. Therefore, it is important to explore techniques that collectively enable reliable GNN training with large graphs on ReRAM-based architectures.

Various methods have been proposed to address the reliability challenges of ReRAM-based architectures due to their low endurance. These techniques range from device-level optimizations to software-level approaches. Device-level methods such as programming latency optimization and low-voltage write schemes have been proposed to improve the reliability of ReRAM crossbars [11, 21]. However, these methods aim at reducing only the per-write energy and do not reduce the actual number of writes. **Structured gradient sparsification (SGS)** and weight **re-mapping (RM)** have been proposed to mitigate the effect of low write endurance [22, 23]. Other endurance-aware approaches such as fault-tolerant training schemes have also been proposed to mitigate the effects of faults and train ML models with limited write endurance [24]. However, these methods focus solely on reducing the number of write operations due to weight updates in **Deep Neural Networks (DNNs)**; they are not directly applicable to GNNs. Unlike DNN training where only weights are updated, GNN training exhibits properties of both DNNs (weight updates), and graph analytics which requires intermittent graph adjacency matrix storage on ReRAM crossbars, thereby leading

to frequent write operations. Moreover, existing fault-tolerant training methods often introduce relatively high area, power, and performance overheads [10, 24].

We show in this work that graph pruning (also known as graph sparsification) can be used to improve device lifetime. Graph pruning is a well-known technique that removes irrelevant edges and nodes in the graph. Traditional graph pruning methods attempt to use unsupervised learning approaches that aim to retain the statistical properties of the original graph. However, these techniques perform poorly on GNN downstream tasks such as node classification or graph classification [25, 26]. Supervised learning-based graph sparsification methods such as NeuralSparse have been proposed to improve generalization by learning to remove from input graphs those edges that are irrelevant for node classification [27]. Also, DropEdge has been proposed as a regularization technique that randomly removes graph edges during training to improve the overall generalization of GNNs. However, as we show later, these methods do not lead to significant benefits in terms of overall performance speed-up, end-to-end latency, and reliability.

Recent work has proposed to generalize the LTH method proposed originally for CNN model weights to GNN models and graphs [12, 13, 15]. These approaches aim to find a pruned GNN model and graph dataset pair that can be trained from scratch to match the accuracy of the unpruned GNN model and dataset. Unified GNN sparsification (UGS, and the **Graph Early Bird Ticket (GEBT)** are two state-of-the art frameworks proposed to prune graphs for GNN training [12, 13]. However, these two methods make use of pruning masks, which are transductive in nature, i.e., they are derived uniquely for a given graph adjacency matrix [3]. Hence, they cannot be transferred or applied to unseen or new graph datasets [3]. Moreover, LTH-based pruning methods are iterative in nature and require many training rounds to prune GNN models and data graphs. As a result, this increases the training cost by more than 20×, and makes it expensive to apply LTH-based pruning to new or unseen large graphs. In addition, these LTH-based GNN pruning methods still aim to prune the entire monolithic graph and they do not reduce the number of input subgraphs used for the end-to-end GNN training process. As we show later, this leads to device reliability issues.

In this work, we address the limitations of existing approaches by proposing a fundamentally different solution based on the concept of learned binary classifiers for low-cost subgraph pruning. This method enhances the reliability of ReRAM architectures by reducing the number of write operations. Subgraph pruning also reduces the training cost and improves the performance of GNN training with minimum accuracy loss.

3 METHODOLOGY

In this section, we first discuss some preliminaries of GNN training, and present the details of the proposed BGC-enabled subgraph pruning framework for GNN training on ReRAM-based manycore architectures (illustrated in Figure 1 and Figure 2). Next, we discuss how BGC enabled pruning improves performance in terms of end-to-end execution time, device reliability, and training cost for a ReRAM-based manycore system during GNN training.

3.1 Preliminaries

A monolithic data graph is represented by an adjacency matrix $\boldsymbol{G} \in \{0,\ 1\}^{n \times n}$, and a node feature matrix $\boldsymbol{X} \in \mathbb{R}^{n \times d}$, where n is the number of nodes in the graph, and d is the dimensionality of each node feature vector. Due to the limited amount of storage available in PIM accelerators, we leverage mini-batch GNN training, where the large monolithic graph (G) is partitioned into smaller parts known as **subgraphs** and used for GNN training [17, 28–30]. Hence, a monolithic data graph G is partitioned into K subgraphs such that $\{SG_1,\ SG_2, \ldots, SG_K\} \subseteq \boldsymbol{G}$, where SG_i denotes the ith subgraph. Each subgraph has its unique adjacency matrix ($\boldsymbol{A}$).

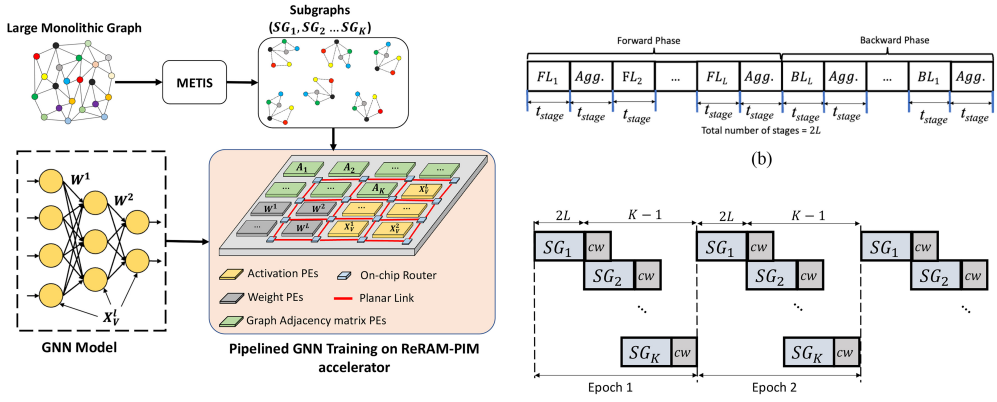

Fig. 1. (a) Overview of GNN training on ReRAM-based PIM accelerator with large monolithic graph partitioned into subgraphs and mapped to PEs, alongside GNN weights, and activations. (b) Pipeline stages during GNN training; each stage is a GNN layer. (c) Pipelined execution of GNN training with K input subgraphs on ReRAM-based accelertors. After each subgraph is processed, crossbar write (cw) operations occur in the form of weights update, intermidate activations storage, and adjacency matrix storage.

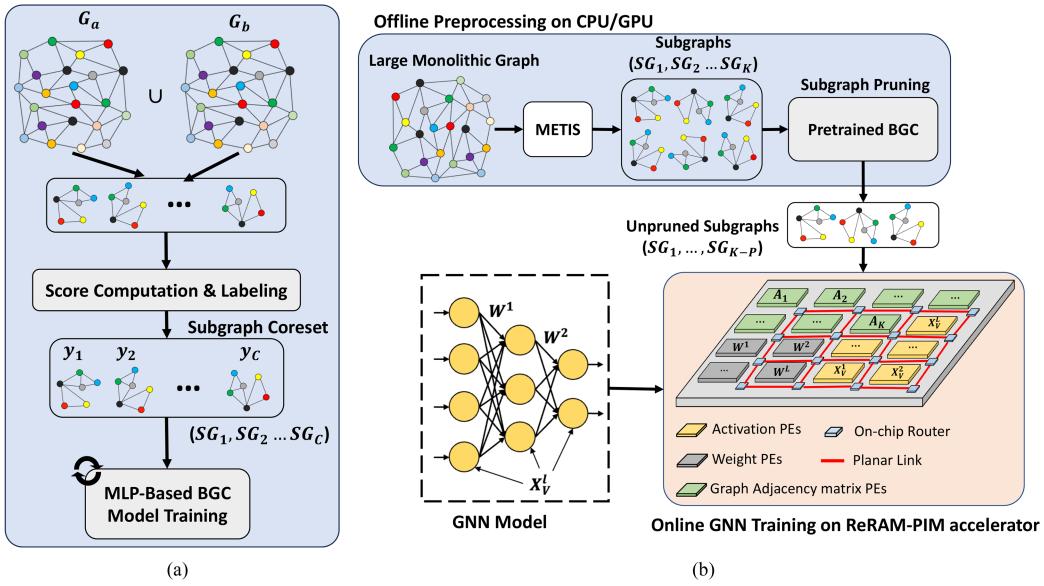

Fig. 2. (a) Overview of training the MLP-based Binary Graph Classification Model. (b) Online GNN training on the ReRAM-based PIM accelerator using the pre-trained BGC model. The training of the MLP-based BGC model is performed offline on a CPU/GPU platform. At the 5^{th} epoch during GNN training, the inference pass of pretrained BGC is performed on the host CPU/GPU device to make decisions about which subgraphs should be pruned. The remaining subgraphs are loaded on the ReRAM-based PIM accelerator for online GNN training.

A GNN consists of L neural layers, where each layer performs forward- and backward-phase computations. The forward-phase computations are described as follows:

Forward-propagation:

$$\text{GNN layer l:} \begin{cases} H^l = X^l \times W^l & (1) \\ H^{l+1} = A \times H^l & (2) \\ X^{l+1} = ReLU\left(H^l\right) & (3) \end{cases}$$

During the forward phase, an MVM operation is performed between the input feature vector (X^l) and the GNN model weights (W^l) for a given GNN layer (l) as shown in Equation (1). The updated feature vector (H^l) is multiplied with the subgraph adjacency matrix (A) to perform the node-feature aggregation of neighboring nodes to produce an output embedding (H^{l+1}). This embedding is passed through a non-linear ReLU activation function to produce the input activations (X^{l+1}) for the next GNN layer ($l + 1$) shown in Equation (3). The backward phase can be expressed as follows:

Backward-propagation:

$$\text{GNN layer l:} \begin{cases} err^l = \left(W^{l-1}\right)^T \times \delta^{l-1} \cdot ReLU'(X^{l-1}) & (4) \\ \delta^l = A \times err^l & (5) \end{cases}$$

Equations (4) and (5) represent the error calculation in the back-propagation phase during GNN training. As shown in Equation (4), the activations of the preceding layer (X^{l-1}) are needed for the computation of the error (err^l) of layer l. Here, $ReLU'$ represents the derivative of the activation function. In Equation (5), this error vector is multiplied with the adjacency matrix to obtain the accumulated error vector (δ^l), which is then used to compute gradients for the weight update process.

3.1.1 GNN Training on ReRAM-based PIM Architectures. ReRAMs enable parallel and efficient in-memory **Matrix-Vector-Multiplication (MVM)** operations. To enable GNN training on ReRAM-based PIM accelerators, the GNN weights are mapped to ReRAM cells in each **processing element (PE)** following [6] and [19]. The graph adjacency matrix of subgraphs, as well as the activations (intermediate output of each GNN layer) are mapped to separate PEs. Figure 1(a) illustrates how the GNN weights, adjacency matrix, and activations are mapped to the ReRAM crossbars. In the forward propagation (Equations (1)–(3)), an MVM operation is performed on ReRAM crossbars in PEs storing the weights between the GNN weights, and the output activations of the previous GNN layer (X^{l-1}). This step is referred to as the 'combination phase' of the forward propagation, as shown in Equation (1). Next, in the 'aggregation phase' during the forward propagation, an MVM operation is performed between the output of the combination phase and the graph adjacency matrix (Equation (2)). To implement these MVM operations on the ReRAM crossbar, one of the operands must be written to the ReRAM cells, while the other operand is fed as an input. In Equations (1) and (2), the weights (W) and the adjacency matrices (A) respectively are written on to the crossbars. Each tile is equipped with a non-linear activation unit to perform the ReLU operation (Equation (3)) following prior work [7, 31]. In the backward propagation, the intermediate activations stored in separate PEs during the forward phase are used to compute the error gradients (err^l). As shown in Equations (4) and (5) above, this operation is simply a MVM operation between the intermediate activations (δ^l), and the transpose of the weights ($(W^{l-1})^T$). Figure 1 illustrates how the GNN weights (W^l), activations (X^{l+1}), and subgraph adjacency matrices (A) are mapped to the ReRAM-based accelerator.

As mentioned earlier, to enable GNN training of large graphs on the ReRAM-based PIM architecture, the large monolithic graph is partitioned into multiple smaller subgraphs ($\{SG_1, SG_2, \ldots, SG_K\}$) using a graph partitioning algorithm [32]. Hence, this allows us to pipeline the GNN training process such that each input **subgraph (SG)** is analogous to an input image to the pipeline as in traditional DNN architectures. Figure 1(b) and Figure 1(c) shows an illustration of the overall pipelined GNN training process. In Figure 1(c), each input subgraph is sent sequentially as input to the pipeline, while Figure 1(b) shows the end-to-end pipeline for training a L-layer deep GNN for one input subgraph. This deep pipelining strategy allows all layers of the GNN to be computed in parallel. As a result, this speeds up the entire GNN training process, and improves throughput and energy-efficiency compared to its unpipelined version. In Figure 1(b), FL_i and BL_i denote the forward- and backward- propagation computation respectively of the lth layer of the GNN during training. The worst-case delay associated with a pipeline stage is denoted as t_{stage}. Following prior work, we assume each GNN layer as one pipeline stage [33]. The training time is determined by the longest latency among all the layers (as shown in Figure 1(b)), and the end-to-end pipeline depth (D) given by:

$$D = 2L + K - 1 \tag{6}$$

Where K is the total number of input subgraphs, and L is number of GNN layers. During GNN training, the overall execution consists of computation (i.e., MVM operations) on ReRAM-based crossbars and inter-PE communication (the data of one layer must be sent to the next layer). The overall execution time is determined by the *end-to-end pipeline depth* (D) and the *delay of each pipeline stage* (t_{stage}). The overall execution time is given by: $D \times t_{stage}$. To improve the execution time, we must reduce the pipeline depth, which is governed by the input dataset, i.e., number of subgraphs (K) used for training.

However, it is well known that multiple writes to the ReRAM cells can lead to device failure [9, 11, 34]. Specifically, during GNN training, ReRAM cell write operations (cw) occur due to weights updates, intermediate activation storage, and subgraph adjacency matrix storage after each subgraph is processed, as shown in Figure 1(c). Moreover, for very large graphs partitioned into many subgraphs (i.e., large K), and for deep GNNs with many layers (i.e., large L), the number of write operations, overall pipeline depth, and end-to-end training latency increases significantly as illustrated in Figure 1(c). Consequently, this leads to severe degradation in the ReRAM reliability (i.e., due to limited write endurance), and incurs significant training performance penalty. From Equation (6) and Figures 1(a)–(c), we can see that reducing the number of subgraph partitions (K), for example, by creating fewer partitions, but larger subgraphs will lead to a reduction in pipeline depth (D) and number of ReRAM cell write operations (cw). However, creating larger subgraphs with high storage requirements for pipelined GNN training on ReRAM-based PIM accelerators is often not desirable. This is due to the small memory footprint of ReRAM-based PIM architectures. Moreover, training GNNs with large subgraphs leads to significant accuracy loss as shown in prior work [33]. Therefore, it is desirable to choose a higher number of partitions with smaller subgraph sizes that can fit within the on-chip memory. However, the high number of write operations due to more subgraphs still leads to reliability issues. Hence, in this work, we propose a unique method for subgraph pruning to reduce all types of write operations to the ReRAM crossbars, thereby improving reliability and the overall training execution time without compromising accuracy.

3.2 Binary Graph Classifier (BGC) for Subgraph Pruning

As discussed earlier, the overall training time and the number of ReRAM crossbar write operations are directly related to the number of input subgraphs for a given graph dataset used for training

GNN models. Consequently, the number of subgraphs has a direct impact on the performance, reliability, and predictive accuracy of GNN model for a given GNN training workload (model and dataset) executed on the ReRAM based manycore platform. The key challenge is to prune the input subgraphs for GNN training on the ReRAM-based PIM architecture, such that the performance in terms of execution time, and reliability is improved without any significant loss in accuracy of the GNN model. To address this challenge, we propose an algorithm to train a binary graph classifier (BGC) for pruning subgraphs on unseen graph data.

We formulate the subgraph pruning problem as a graph classification task. Given a set of subgraphs $\{SG_1, SG_2, \ldots, SG_K\}$ for a graph dataset G, a trained subgraph classifier will predict a binary label for a given subgraph (SG_i): 0 means that the subgraph should be pruned and 1 means that the subgraph should be retained for training the given GNN model. We employ the subgraph representation from a trained GNN model and use a simple MLP classifier on top of this learned subgraph representation, which is obtained by aggregating the node features of the subgraph using a graph-level pooling function [30]. The trained subgraph pruning classifier (BGC) produces a binary output prediction $\hat{y}_i$ (0 means to prune and 1 to mean not prune) for a given subgraph SG_i. The key insight behind pruning the subgraphs is to exploit the inherent redundancy of information in the graph including nodes with similar features, labels, and neighborhood. Additionally, with a small number of training iterations, it is possible to use the GNN model to score the subgraphs in terms of their usefulness for further training [33, 35]: a high score for a subgraph means that the GNN model can make accurate predictions (node/edge classification) and additional training using such subgraphs will not improve the predictive accuracy of the GNN model. We leverage these insights to design an effective algorithm to train such subgraph pruning classifiers. Note that we perform subgraph pruning using this trained classifier after a small number of initial training epochs. We demonstrate later that pruning subgraphs after five training epochs gives consistently good results on diverse training/testing splits over a set of GNN training workloads. There are two key challenges in creating an effective binary classifier for subgraph pruning on unseen graph data: (1) How to create supervised training data of subgraphs and binary pruning decision pairs? (2) How to train a binary classifier on a given imbalanced training dataset where the number of subgraphs with 'prune' labels will be higher than those with 'not prune' labels. In what follows, we provide algorithmic solutions to answer these two questions.

Creation of Supervised Binary Classification Data. We assume the availability of a set of known GNN training workloads (i.e., GNN model and graph data pairs: $\{gnn_i, G_i\}$) for this purpose. Suppose we use a fixed early training epoch E (say 5th iteration) at which we will perform subgraph pruning for any given GNN training workload. For a given graph $\boldsymbol{G_i}$ there can be many subsets of subgraphs that can preserve the GNN model accuracy after training epoch E. Algorithm 1 describes the overall supervised binary classification data generation process. Ideally, we want to find the smallest subset of accuracy preserving subgraphs ($\{SG_1, \ldots, SG_C\}$) and the corresponding pruning labels ($\{y_1, y_2, \ldots, y_C\}$) because it aligns with our overall goal. However, this leads to a hard combinatorial optimization problem (search over subsets of different sizes in the decreasing order and performing GNN training with each candidate subset). Additionally, there can be multiple different subsets with the same minimum size and selecting one of them arbitrarily for different GNN training workloads can create inconsistent training data for subgraph pruning classifier. To overcome these challenges, we leverage recent work on DNN model-based importance scoring of input examples for data pruning [35] and generalize this to subgraph pruning for a pre-specified pruning percentage. The importance score assigned to each subgraph corresponds to the likelihood of the subgraph can be pruned: a low score means the corresponding subgraph can be pruned with a high probability (lines 5–9 of Algorithm 1). The key idea behind our approach is to create binary classification data as follows. First, we use these importance scores to sort the subgraphs. Second,

ALGORITHM 1: Creation of Supervised Binary Classification Data

Input : GNN training workloads: GNN + Graph Set $\{gnn_i + G_i\}$
Output: BGC Training data and labels: $\{SG_1, ..., SG_C\}$, $\{y_1, ..., y_C\}$,
1 **Initialize** GNN;
2 **for** *each* gnn_i *and* G_i **do**
3 **while** *No accuracy loss in* gnn_i **do**
4 **Train** gnn_i with G_i for E epochs ;
5 **Compute Score:** $score(SG_j) = \sum |\delta^l(SG_j)W_i| \; \forall \; \mathrm{SG}_j \subset G_i$;
6 **Sort:** subgraphs as-per *scores*;
7 **if** $E == 5$ **then**
8 **Prune:** subgraphs with low scores
9 **end if**
10 **Search:** max value of k, such that removing k% subgraphs preserves ideal accuracy;
11 **Label:** subgraphs with top-k% scores as '1';
12 **Label:** remaining subgraphs as '0'
13 **end while**
14 **end for**
15 **Aggregate** subgraphs and labels from all GNNs;
16 **Return**$\{SG_1, ..., SG_C\}$ and **labels**: $\{y_1, ..., y_C\}$;

we train the GNN model by pruning the top k percent subgraphs for different values of k in an increasing order to find k such that the accuracy is preserved with respect to the ideal setting (no subgraphs are pruned). By gradually increasing the value of k (in discrete intervals, say 10) until we see loss in accuracy (line 10 of Algorithm 1), we identify the maximum subset of subgraphs that can be pruned in a sound manner. Since this experiment with each value of k involves training the GNN model, we perform a discrete search to efficiently identify the maximum value of k as part of a one-time offline process. Importantly, sorting the subgraphs based on their importance scores injects inductive bias to avoid the generation of inconsistent binary classification data across different GNN training workloads. We aggregate the binary classification data over all GNN training workloads to form our overall training set for the binary subgraph classifier.

Model-based Importance Scoring: We score each subgraph (SG_j) in the set of subgraphs, as shown in Equation (7) and line 5 of Algorithm 1. This formulation, described by Equation (7), scores the importance of each subgraph (SG_j) by its expected loss gradient norm, which bounds (up to a constant) the expected change in loss for an arbitrary subgraph example caused by removing SG_j [35]. In other words, subgraphs with low scores are expected to have a limited influence on learning how to make predictions (based on their structure) for the rest of the training subgraphs. The importance of each subgraph in the coreset (SG_j) is determined by a scoring function ($score(SG_j)$) described below:

$$score(SG_j) = \sum |\delta^l(SG_j)\Delta W_l| \tag{7}$$

We have conducted experiments where the above scoring function was used to rank all subgraphs based on the magnitude of the product of the gradient with respect to the input subgraph SG_j (i.e., $\delta^l(SG_j)$) and the change in the weights in GNN layers (ΔW_l) [33]. This score indicates the contribution of each subgraph (SG_j) to the change in the overall training loss function. The parameter δ^l is the gradient update, like the one used earlier in Equation (5). We normalize the importance score associated with each subgraph SG_j to a continuous value ranging from 0.0–1.0 to account for variability across different GNN training workloads. Hence, to generate the corresponding labels of each subgraph ($\{y_1, y_2, \ldots, y_C\}$), we quantize the score values to be binary values (0 or 1) by identifying the maximum top k% subgraphs from the sorted list based on the

importance scores that can be pruned without any loss in accuracy. This is done by searching over discrete values of k and performing GNN training with top-k% pruned subgraphs. In this work, the value of k was chosen to be 10, consistent with prior work [33].

Training Binary Subgraph Classifier from Imbalanced Data. It is conceivable, and empirically observed in both prior work and current work, that the number of subgraphs that can be pruned are higher (majority class) than those that cannot be pruned (minority class) for each GNN training workload, even though the specific imbalance ratio varies from one training workload to another. Additionally, the accuracy of the trained subgraph pruning classifiers can impact performance, power, reliability, and model accuracy when the GNN is trained on a ReRAM-based hardware platform. For example, if we prune important subgraphs (determined by Equation (7)), then we can potentially lose GNN model accuracy. Similarly, if we do not prune unimportant subgraphs, then we incur penalties in terms of performance, power consumption, and reliability.

A naïve approach of using standard classification training data will result in a classifier that will be biased to predict the majority class; however, this approach potentially leads to a significant loss in predictive accuracy. To overcome this challenge, we propose to use a cost-sensitive classifier training approach that assigns higher importance weight to examples from the minority class as we do not want to make mistakes on those examples. Algorithm 2 demonstrates the overall training process of the MLP-based BGC using a weighted binary cross entropy loss function ($\mathcal{L}_{wght}$) shown in Equation (8), and line 5 of Algorithm 2. A weighted cross entropy loss function is used to prevent any bias in the BGC model that may arise due to class imbalance in the overall binary classification training data. The weighted binary cross entropy loss is given by:

$$\mathcal{L}_{wght} = -\frac{1}{C}\sum_{j=1}^{C}\left[w_p \cdot y_j . \log\left(\hat{y}_j\right) + \left(1 - y_j\right) . \log\left(1 - \hat{y}_j\right)\right] \tag{8}$$

The weighted cross entropy loss applies a rescaling parameter (w_p) to the conventional binary cross entropy loss [36], allowing us to penalize false positives or false negatives appropriately depending on the needs of the application at hand. This **cost-sensitive learning (CSL)** process ensures that the BGC model pays more attention to the minority class, thus improving its performance in cases when the training set is imbalanced. Here, y_j and $\hat{y}_j$ are the binary label and prediction from the MLP-based binary classifier (f_{MLP}) model, respectively, of a given subgraph (SG_j) in the BGC training set (line 4 of Algorithm 2), and C is the total number of subgraphs in the training set. As we show later in our experiments, we automatically select the value of w_p based on the performance on a small validation data set such that the BGC model is trained to make unbiased predictions when used for the online training process on a ReRAM-based PIM accelerator. In summary, using the training subgraphs ($\{SG_1, SG_2, \ldots, SG_C\}$) as training data for the MLP-based BGC model, we have described how a classifier can be trained to accurately predict if a new subgraph should be pruned or not for an unseen GNN training workload. We refer to this overall framework as the Binary graph classifier (BGC), where the MLP-based model is pre-trained offline and used to prune subgraphs of any given dataset during online GNN training on the ReRAM-based PIM accelerator. In this context, 'offline' refers to any training process that is carried out on conventional hardware platforms, while 'online' refers to training executed on the ReRAM-based PIM accelerator.

The BGC model is a light-weight MLP-based neural network (consisting of only few layers), which is pre-trained to reliably perform the subgraph pruning task for unseen GNN training workloads. The goal of training the BGC model is to learn the subgraph network structure, and not to learn subgraph node-level representation as in the actual GNN training task. Hence, to train the BGC model we employ the aggregate set of subgraphs ($\{SG_1, \ldots, SG_C\}$) from the known GNN

ALGORITHM 2: Offline Binary Graph Classifier Training

Input : BGC training subgraphs; $\mathcal{G} = \{SG_1, ..., SG_C\}$, Training labels $\mathcal{Y} = \{y_1, ..., y_C\} \mid \mathcal{Y} \in \{0, 1\}$, MLP binary classifier $f_{MLP}(\mathcal{G}|\mathcal{Y}, W_{MLP})$

Output: Trained MLP classifier f^*_{MLP}

1 **Initialize** f_{MLP};
2 **for** *each training iter* **do**
3 **for** *each* $SG_j \in \mathcal{G}$ **do**
4 Make prediction $\hat{y}_j = f_{MLP}(SG_j)$;
5 $\mathcal{L}_{wght} = -1/C \sum [w_p \cdot y_j \log(\hat{y}_j) + (1 - y_j) \log(1 - \hat{y}_j)]$;
6 Compute gradients in backward-pass ;
7 Update f_{MLP} weights (W_{MLP})
8 **end for**
9 **end for**
10 **Return** f^*_{MLP}

training workloads along with their binary labels. Note that subgraphs ($\{SG^U_1, \ldots, SG^U_K\}$) refer to an unknown GNN training workload and will be used for the online GNN training process on the ReRAM-based PIM accelerator. Moreover, the training of the MLP-based BGC model using the training subgraphs is a one-time event performed offline on conventional architectures (such as CPU/GPU). The pre-trained BGC model can be used multiple times to prune subgraphs for GNN training with any given graph dataset on the ReRAM-based PIM accelerator.

3.3 GNN Training with Unseen Graphs on ReRAM-based Accelerators using Subgraph Pruning Classifier

In Figure 2, we present a high-level illustration of how the BGC model is pre-trained offline, and then used for online GNN training with an unseen data graph on the PIM architecture. In this subsection, we explain how the pre-trained MLP-based BGC model is used for subgraph pruning during the online GNN training process on the ReRAM-based PIM architecture. Algorithm 3 describes the online subgraph pruning process, for new or unseen graph datasets. It has been demonstrated in recent work that redundant examples can be pruned very early in training, without significant degradation in the final test accuracy [13, 35]. Hence, in this work, we propose to apply the pre-trained BGC model to prune training examples (subgraphs in this case) very early in the GNN training process (*within a few epochs*). In other words, the pre-trained BGC model is used to prune the input training samples. For our analysis, we consider a scenario where the BGC-enabled subgraph pruning is applied before the start of training ($E = 0$), and very early during the GNN training process ($E = 5$). As we show in our experimental results, pruning subgraphs at the fifth epoch (line 3 of Algorithm 3) is sufficient to accelerate the GNN training process, and yield significant improvement in reliability and performance of the ReRAM-based PIM accelerator with negligible loss in accuracy. The pretrained BGC model is a light-weight MLP-based neural network consisting of only a few layers and parameters. It is being executed offline on the CPU/GPU host as shown in Figure 2(a). Hence, executing the pretrained BGC model to prune input subgraphs during training is a simple inference pass of the MLP layers, which has a negligible timing overhead. Moreover, this process is performed only once very early during a given GNN training workload instance. Consequently, as shown in Figure 2(b), P subgraphs have been predicted by the BGC model as unimportant and are pruned (line 5 of Algorithm 3). The remaining important subgraphs ($\{SG^U_1, \ldots, SG^U_{K-P}\}$) of the new/unseen dataset are used for online GNN training on the

Table 1. BGC Model and Dataset Specs

Layer Type	#Layers	# parameters	Size	Lr	Epochs
FC (2), ReLU (1)	3	66K	0.13MB	0.01	50

ALGORITHM 3: Online BGC-enabled Subgraph Pruning

Input : Pre-trained BGC: f^{*}_{MLP}; Unknown Graph dataset: $\mathcal{G}_{new} = \{SG^{U}_{1}, ..., SG^{U}_{K}\}$; GNN Model: $g(\mathcal{G}_{new}, W)$
Output: Unpruned Subgraphs $\{SG^{U}_{1}, ..., SG^{U}_{K-P}\}$
Initialize $g(\mathcal{G}_{new}, W)$;
while *GNN training for E epochs* **do**
  if $E == 5$ **then**
    Apply $f^{*}_{MLP}(G_{new})$;
    P subgraphs are pruned;
  end if
end while
Return $(\{SG^{U}_{1}, ..., SG^{U}_{K-P}\})$

ReRAM-based architecture. Instead of training on K subgraphs, the GNN model now trains only on $(K - P)$ subgraphs. Consequently, this reduces the pipeline depth as illustrated in Figure 1(b) and Equation (6), which automatically speeds up training as there are fewer input subgraphs (hence smaller end-to-end pipeline depth (D)). As we show later, this process leads to trained GNN models with little to no test accuracy loss when compared to training with all subgraphs (no subgraph pruning setting). In Section 4, we show the predictive performance of the BGC-enabled pruning for GNN models at different early epochs during the online training process. Overall, this leads to a reduction in ReRAM crossbar cell write operations, thus enabling reliable GNN training on the ReRAM-based PIM architecture.

4 EXPERIMENTAL RESULTS AND ANALYSIS

In this section, we evaluate the performance of the proposed subgraph pruning framework for training diverse GNN models with large real-world graphs on ReRAM-based PIM architectures.

4.1 Experimental Setup

Datasets and GNN models (Workloads): We consider three types of state-of-the-art GNN models to demonstrate generality of our proposed subgraph pruning approach, namely: **Graph Convolution Network (GCN), Graph Attention Networks (GAT)**, and **Graph Sample and Aggregate (SAGE)**. The proposed BGC technique leverages a simple MLP-based Neural Network (also known as a feed-forward network) for pruning subgraphs during GNN training. Table 1 shows the BGC model based on a multilayer perceptron network and its training configuration. The BGC model is trained offline on the host machine (CPU/GPU) for 50 epochs, with a learning rate of 0.01. In this work we evaluate the performance of the BGC technique on a total of six graphs ranging from different application domains like: medicine (PPI), social-networks (FlickR, Reddit and Yelp), e-commerce (Amazon2M), and a citation dataset (ogbl-citations2). The datasets include small- (consisting of <100K nodes and edges) to large-scale (consisting of >1M nodes and edges) graph datasets in this work [29]. Hence, they are representative of a wide range of real-world applications of predictive graph analytics specifically suitable for edge AI applications. In this paper, we have shown that the BGC method can be used to make pruning decisions for different datasets (small to large). This demonstrates the scalability and transferability of the BGC

Table 2. GNN Configuration

Model	#layers (L)	lr	Epochs
GCN	4	0.01	200
SAGE	4	0.01	200
GAT	4	0.005	200

Table 3. Graph Dataset Statistics

Dataset	#Nodes	#Edges	Classes	Node degree	K
PPI	56K	818K	121	15	50
Reddit	233K	11.2M	41	50.5	150
Amazon2M	2.45M	61.9M	47	10	1000
FlickR	89K	899K	7	10	75
Yelp	716K	13.9M	100	10	500
Ogbl-citation2	2.9M	30.5M	6	20.7	1500

Gray shaded datasets are used for training the BGC.

framework to larger graph workloads from diverse application domains without requiring any additional training. Tables 2 and 3 provide the details of the GNN models and the graph datasets respectively used in this work. All GNN models considered here utilize mini-batch based training, which uses multiple smaller subgraphs created by using a graph partitioning algorithm from a large monolithic graph. Different graph partitioning algorithms have already been previously explored extensively in the literature, such as Random Edge-cut algorithm, METIS, **Degree-based hashing (DBH)**, and edge partitioning via **Neighborhood heuristic (NH)** [37, 38]. A recent ablation study on partition algorithms for GNN training showed that METIS achieves superior performance in terms of accuracy compared to other partitioning algorithms [38]. Moreover, unlike the other methods, METIS creates graph partitions such that each subgraph can be bounded by a controlled size, so that a batch of a fixed number of subgraphs can always fit within the on-chip memory [32, 37]. This makes METIS amenable to resource constrained platforms such as ReRAM-based **Processing-in-Memory (PIM)** architectures. Hence, in this paper, we specifically use the METIS graph partitioning algorithm. However, it is worth noting that the proposed BGC framework in this work can be used with other graph partitioning algorithms.

Moreover, this enables pipelined training of GNNs [19]. For graph partitioning, we employ METIS, which is a fast and scalable graph clustering algorithm [32]. The graph clustering algorithm only takes a small portion of preprocessing time and allows us to train GNNs with large datasets. The graph partitions (subgraphs) are loaded from main memory onto the ReRAM architecture for the online training process. As shown in Table 2, each GNN model is trained for 200 epochs (one training instance) with suitable learning rates (lr) and appropriate number of GNN layers (L) to ensure their convergence without overfitting. The number of input subgraphs (K) obtained using the METIS graph partitioning algorithm for each dataset is shown in Table 3. Following prior work, we chose the number of subgraph partitions (K) appropriately for each graph dataset to achieve the best possible trade-off between the GNN accuracy and performance [15, 33].

ReRAM-based PIM Architecture: The PIM architecture considered (without loss of generality) in this work consists of 36 ReRAM-based processing elements (PEs) distributed over four planar tiers connected using through silicon via (TSV)-based vertical links. Table 4 presents the

Table 4. Hardware Specifications

Offline Preprocessing Hardware:	OS: Linux CentOS. GPU: NVIDIA V100, 32GB. CPU: Intel Xeon Gold 5222 @ 3.8GHz, 16 cores, 32K L1 cache and 1024K L2 cache.
Online in-field Training Hardware:	ReRAM-based 3D PIM architecture: 4 planar tiers, 9 PEs per tier, 4 ReRAM tiles per PE.
ReRAM Tile:	96-ADCs (8-bits), 12 × 128 × 8 DACs (1-bit), 96 crossbars, 96 SRUs, 128 × 128 center crossbar size, 10MHz, 2-bit/cell resolution, 0.34W, 0.38 mm^2 [7].

details of the ReRAM-based PIM architecture used in this work. We model the area, energy, and latency of all on-chip buffers, peripheral circuits, number of writes to ReRAM cells, and GNN training accuracy using the NeuroSim v2.1 simulator [31]. We incorporate a Python wrapper function based on PyTorch API into NeuroSIM to obtain the GNN model test accuracies for the different GNN models and datasets considered in this work. The GNN model weights and activations are stored using 16-bit fixed point precision on the ReRAM crossbar of size 128 × 128, and 2-bit/cell resolution as it provides the best performance-area-energy trade-off [19]. The adjacency matrix is binary ('1' indicates the presence of an edge between two nodes and '0' indicates no connection). Hence, it is stored with 1-bit/cell. We incorporate **stochastic rounding units (SRUs)** in each tile to reduce the accuracy drop due to the use of 16-bit precision [39]. Stochastic rounding enables high accuracy even at lower precision settings [40]. The implementation of the stochastic rounding unit introduces less than 1% area and energy overhead. GNN training generates an enormous volume of traffic, which can be a bottleneck performance [6, 19]. In this architecture, we utilize a multicast-enabled 3D mesh **network on chip (NoC)** as the interconnection backbone for communicating between the processing elements during GNN training [39]. The NoC performance has been modeled using Garnet, and NeuroSim was used to calculate the injection rate needed for the cycle-accurate NoC simulator [41].

4.2 Accuracy of the BGC

In this subsection, we discuss the performance of the MLP-based BGC for pruning subgraphs during GNN training. As shown in Table 1, the BGC model is a light-weight model (only 0.13 MB of storage). In comparison, a GCN model considered in this work trained with the PPI dataset consists of ~13 M parameters and requires 49.9 MB of storage. Hence, the MLP-based BGC introduces negligible overhead in terms of storage and training time cost as we show later. The BGC comprises of only two **fully connected (FC)** layers, and one ReLU non-linear activation layer. As discussed earlier, we train the BGC model with a core set of subgraphs ($\{SG_1, SG_2, \ldots, SG_C\}$) drawn from two known graph datasets (G_1 and G_2 as an example). Subgraphs drawn from different known graph datasets usually differ in input feature dimensions. Hence, to enable training the BGC with these coreset of subgraphs, we pad the input feature vectors of subgraphs in the core set ($\{SG_1, SG_2, \ldots, SG_C\}$) to match the input dimension of the first FC layer of the BGC model. We then use the trained BGC model to prune subgraphs for new (or unseen) datasets during online GNN training on the ReRAM-based PIM architecture. However, we must first evaluate the performance of the BGC in terms of its classification accuracy before it is deployed for in-field GNN training scenarios with new datasets. Hence, we use the **leave-one-out cross-validation (LOOCV)** procedure to estimate the performance of the BGC model when it is used to make predictions on graph datasets not used to train the model. Here, we consider a third dataset (G_3) as the validation graph dataset used in the LOOCV procedure. We then perform LOOCV experiments with the BGC using G_1, G_2, and G_3. The LOOCV experiments yield three cases where the BGC is trained with: (i) G_1 and G_2, and validated with G_3, (ii) G_2 and G_3, and validated with G_1, and (iii) G_1

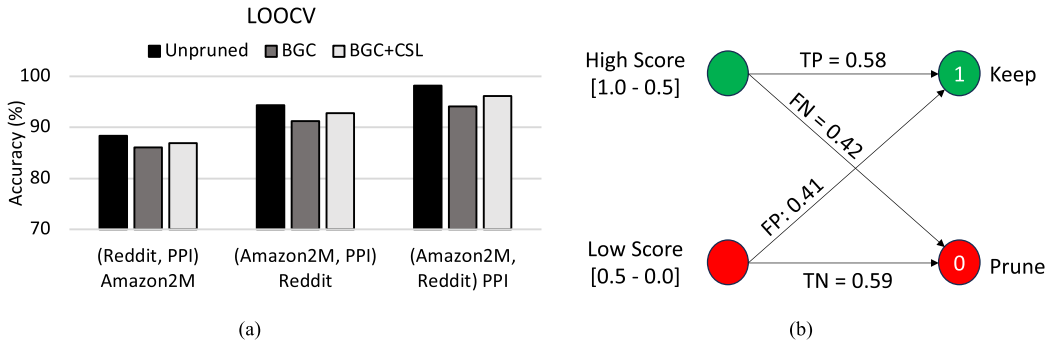

Fig. 3. (a) Leave-one-out cross-validation (LOOCV) experiments for the BGC using Reddit, Amazon2M, and PPI. The BGC is trained with dataset in parentheses, and used for subgraph pruning during GNN training on the validation dataset. We compare GNN training with the unpruned data graph, with the BGC enabled subgraph pruning, and the BGC trained using a weighted loss function. (b) An illustration of True Positive (TP), True Negative(TN), False Postive (FP), and False Negative (FN) rates for the trained BGC making predictions on the PPI dataset.

and G_3, and validated with G_1. LOOCV experiments tend to be computationally expensive as the number of datapoints increase [42]. Hence, in this work, we limit the number of datapoints used to only three datasets (G_1, G_2, and G_3). The overall performance of the BGC model is estimated in terms of accuracy when used to make pruning predictions during GNN training on the validation dataset as demonstrated in the LOOCV experiments.

In Figure 3(a), we present the results of the LOOCV experiments performed on the BGC. We consider PPI Reddit, and Amazon2M as the representative datasets (G_1, G_2, and G_3) used in evaluating the BGC model. Note, however, that the BGC method is general, and can be applied to any other graph datasets. The datasets used to train the BGC in each case are shown in parentheses, followed by the dataset used for validation. For example, when the BGC is trained with Reddit and PPI, and validated using Amazon2M, we denote this as "(Reddit, PPI) Amazon2M". Here, for each case in the LOOCV experiment, we train the BGC with two datasets, and use then use it for making pruning predictions on the validation dataset during GNN training. As shown in Figure 3(a), we compare the accuracy of a GNN trained using the entire validation dataset (unpruned), with the accuracy of the GNN trained with the BGC-enabled subgraph pruning on the validation dataset (BGC). For the LOOCV experiments, we consider the GCN model as the representative GNN trained with the validation dataset in each case. Our results demonstrate that the GCN can be trained with BGC-enabled data pruning with an average accuracy loss of ~4.0% in all three scenarios as shown in Figure 3(a). Note that we observe a similar accuracy loss trend when the LOOCV experiments are performed using other GNN models (GAT and SAGE). This relatively high average accuracy loss of 4.0% can be attributed to the false predictions made by the BGC model. Hence, we quantify the **false positive (FP)** and **false negative (FN)** prediction rates when PPI (as an example) is used as the validation dataset (i.e., the (Amazon2M, Reddit) PPI case), as shown in Figure 3(b). In this context, an FN prediction means pruning an important subgraph with a high score that should otherwise be kept, and an FP prediction implies keeping a subgraph with a low score that should otherwise be pruned. On the other hand, **true positive (TP)** and **true negative (TN)** rates denote the accurate subgraph pruning predictions by the BGC. We observe relatively high FP and FN prediction rates of 0.41 and 0.42 respectively for the BGC when it is used to prune the PPI dataset. This implies that the BGC is making subgraph pruning predictions with a high degree of uncertainty. As a result, this can potentially have negative effects on the

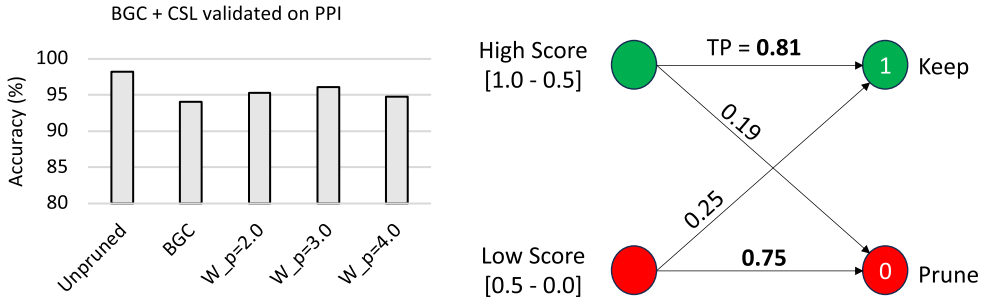

Fig. 4. (a) Ablation experiments used to chose the value of importance weighting parameter w_p to train the BGC model using a cost-sensitive training approach (CSL). Here, the BGC is trained using Amazon2M and Reddit dataset, and tested with the PPI dataset. (b) An illustration of True positive (TP), True negative (TN), False postive (FP) and False negative (FN) rates for the improved BGC trained using the weighted loss function, and used to make predictions on PPI.

performance and reliability of the ReRAM-based PIM architecture. This also impacts the accuracy of the GNN model. The goal of the BGC model is to prune unimportant subgraphs with low scores and keep important subgraphs with high scores as shown in Figure 3(b). Hence, this implies that the BGC model should have high true positive (TP) and true negative (TN) rates, and low FN and FP rates, which is not the case as shown in Figure 3(b). Hence, we address this challenge of high false prediction rates by proposing a **cost-sensitive learning (CSL)** method for the BGC using a weighted loss function, as introduced in Section 3.

Training the BGC model using weighted loss: To address the accuracy loss of the BGC-enabled pruning due to its high FP and FN rates (shown in Figure 3(b)), we propose to train the BGC using a cost-sensitive learning approach discussed earlier. This cost sensitive learning approach uses a weighted binary cross entropy loss function (bce_{wght}) to train the BGC model while penalizing false positives or false negatives significantly. This is achieved by using a rescaling parameter w_p as shown in Equation (8). We perform ablation experiments shown in Figure 4(a) to find a suitable value for w_p. In Figure 4(a), the BGC is trained with the Amazon2M and Reddit datasets using the cost-sensitive loss ($\mathcal{L}_{wght}$) in Equation (8) and is validated on the PPI dataset. As shown in Figure 4(a), our experiments demonstrate that a $w_p = 3.0$ is sufficient to improve the performance of the BGC in-terms of accuracy when used to prune the validation dataset. However, although Figure 4(a) shows the (Reddit, Amazon2M) PPI case, we generally observe that a $w_p = 3.0$ works well in other scenarios (i.e., (Reddit, PPI) Amazon2M) and (Amazon2M, PPI) Reddit). Hence, in our LOOCV experiments shown in Figure 4(a), we fix the $w_p = 3.0$, and train the BGC using the cost-sensitive learning approach (BGC+CSL) as discussed in Section 3. Consequently, we now observe a GNN average accuracy loss of only 2.1% compared to GNN training with the entire dataset (unpruned). In Figure 4(b), we also observe a corresponding reduction in the FP and FN rates to 0.25 and 0.19 respectively, and an improvement in the TP and TN rates to 0.81 and 0.75 respectively for PPI. Note that we also observe a similar trend with Reddit, and Amazon2M when they are also used as the validation dataset. Overall, our LOOCV experiments have given a better insight to evaluating the performance of the BGC model. For the rest of our experiments and analysis, we use a the BGC trained with Reddit on PPI dataset (highlighted in gray in Table 3). We then carry out our performance, reliability, and accuracy trade-off analysis for the BGC-enabled GNN training on the ReRAM based PIM architectures using other datasets (or new datasets).

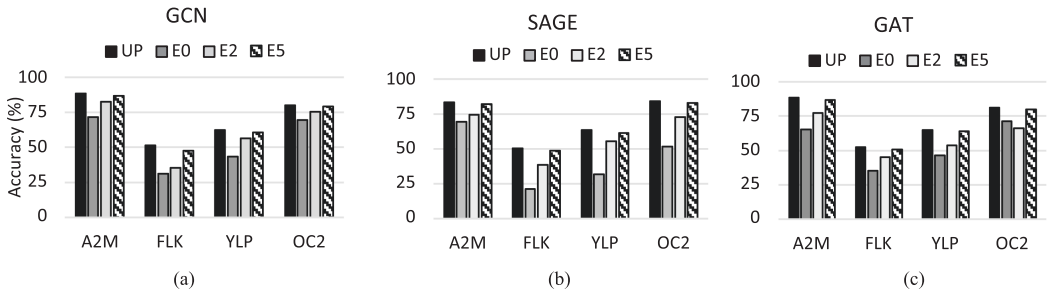

Fig. 5. (a)–(c) Test accuracy of of the unpruned (UP) GNN model for Amazon2M (A2M), Flickr (FLK), Yelp (YLP), and ogbl-citation2 (OC2) compared to BGC enabled subgraph pruning applied; before training (E0), at epoch 2 (E2), and at epoch 5 (E5) for GCN, SAGE, and GAT models.

4.3 BGC-enabled Subgraph Pruning for GNN Training

So far, we have introduced the BGC model, and demonstrated its ability to prune subgraphs of a graph dataset during GNN training. In this subsection, we present the results of executing the BGC for subgraph pruning during GNN training with new or unseen datasets. To do this, we employ four datasets as shown in Table 3, namely Amazon2M, Flickr, Yelp, and ogbl-citation2. The datasets considered here were not used in training the BGC, hence they are referred to as new or unseen graph datasets. We consider PPI and Reddit as 'known' datasets because they were used to train the BGC. Hence, for the rest of our analysis, we evaluate the performance of the BGC only on the unseen graph datasets: Amazon2M, Flickr, Yelp and ogbl-citation2. During the execution of the GNN training on the ReRAM-based PIM architecture, the pre-trained BGC is used to perform the subgraph pruning. When the BGC model is executed, it predicts P subgraphs to be unimportant, and then prunes them. As a result, this leaves behind $K - P$ subgraphs (i.e., $\{SG_1, \ldots, SG_{K-P}\}$), which are then used for the GNN training process. The remaining or unpruned subgraphs are then loaded on the ReRAM-based architecture and used for the rest of the GNN training process.

However, determining when to prune remains crucial to enabling reliable GNN training with significant performance speedup. For example, pruning datasets early (within few initial epochs) during training reduces the total number of write operations to the ReRAM crossbar cells. As a result, this potentially allows more training instances to be executed on the ReRAM crossbar before it reaches its endurance limit. In addition, reducing the number of subgraphs used for training shortens the pipeline depth (D), thereby leading to performance speed-up of the entire GNN training process. Recent work has shown that input data can be pruned very early during the training process, without significant accuracy loss [13]. In Figures 5(a)–(c), we present the final GNN test accuracy when the BGC-enabled subgraph pruning is applied at different points within the first five epochs. Here, the unpruned case refers to the GNN training on the ReRAM-based architecture using all subgraphs without the BGC-enabled pruning. We apply the BGC subgraph pruning before the start of training (E0), at the second epoch (E2), and at the fifth epoch (E5). We perform these experiments on the GCN, SAGE, and GAT models and on the Amazon2M, Flickr, Yelp, and ogbl-citation2 graph datasets. As shown in Figures 5(a)–(c), our results demonstrate that when the BGC-enabled subgraph pruning is performed at the fifth epoch, this leads to a minimal accuracy loss of compared to at other epochs. For example, in GCN, applying the BGC at Epoch 5 leads to ∼2% accuracy loss on average across all datasets. Meanwhile, applying the BGC before the start of training (E0), and at epoch 2 (E2) leads to ∼17% and 9% accuracy loss on average respectively compared to the unpruned case. Applying the BGC subgraph pruning too early (for example, at E0) can potentially lead to significant accuracy loss, as the model suffers from information loss during

Table 5. Data Pruning Percentage (p) Achieved with Different GNN Models and Datasets at Epoch 5

Dataset	Amazon2M		FlickR		Yelp		Ogbl-citation2	
Pruning Method	BGC	SDP	BGC	SDP	BGC	SDP	BGC	SDP
GCN	43.2%	45.0%	24.50%	26.00%	33.2%	36.0%	40.31%	45.0%
SAGE	41.0%	44.0%	29.5%	32.0%	30.5%	35.0%	39.50%	40.0%
GAT	45.1%	44.0%	30.2%	33.3%	31.51%	35.0%	38.33%	40.0%

training. Hence, we propose to apply the BGC-enabled subgraph pruning at the fifth epoch (E5) during GNN training in a one-shot manner. Importantly, the training data to create BGC model is generated in a consistent manner for E5. The remaining unpruned subgraphs are then used for the rest of the GNN training process, hence leading to an improvement in the overall training performance speed-up without a significant loss in the GNN test accuracy. Overall, we observe a similar trend with the SAGE and GAT models in Figure 5(b) and Figure 5(c) respectively across all datasets. In Table 5, we show the percentage of data pruning achieved by the BGC at the fifth epoch. As discussed in the previous subsection, when P subgraphs are pruned, the percentage (p) of subgraph pruning achieved is; $p = \frac{(K-P)}{K} \times 100\%$, where K is the total number of subgraphs (as shown in Table 3). To benchmark the performance of the proposed BGC classifier in terms of its achievable pruning percentage, we compare the pruning percentage (p) of the BGC with a recently proposed **score-based data pruning (SDP)** approach as a baseline (shown in Table 5) [33]. As shown in Table 5, we observe that the SDP approach achieves a slightly higher pruning percentage of 3.5% on average for each dataset. The SDP method initially trains the GNN model on the unseen dataset to directly find unimportant input subgraphs using the GNN training loss function and prunes them. However, as we show later in our experimental results, this approach requires multiple training rounds to find the appropriate pruning percentage, which significantly raises the training time cost. Moreover, this approach is not generalizable to unseen graph datasets without requiring prior training on each new graph dataset. The aim of the BGC method is to also achieve high pruning percentages similar to the SDP approach, without prior GNN training iterations with the unseen graph to find important subgraphs. As we show in the next sub-section, the pretrained BGC method requires no prior, hence outperforms the SDP approach in-terms of overall training time cost. We observe that large graphs with higher number of nodes, edges, and subgraphs (K) such as Amazon2M and Ogbl-citation2 (as shown in Table 3) tend to achieve a higher amount of pruning relative to FlickR and Yelp as we see in Table 5. This can be attributed to the higher amount of redundant information present in larger graphs. Hence, they can be pruned more during GNN training without incurring significant accuracy loss.

Training time speedup: As discussed above, pruning subgraphs using the BGC at the fifth epoch (E5) leads to less accuracy loss compared to earlier epochs. Hence, in this work, we compare the GNN training performance speedup achieved on the full dataset (unpruned) with our proposed BGC-enabled subgraph pruning applied at the fifth epoch (E5) during GNN. Figure 6(a), Figure 7(b), and Figure 7(c) show the performance speedup achieved by the BGC-enabled subgraph pruning for GCN, SAGE, and GAT, respectively. The BGC is a light-weight MLP model consisting of only three layers. Hence, a forward pass on the pretrained BGC takes less than 1% timing overhead of the overall training execution time. This negligible time overhead (1%) required for the pretrained BGC forward pass is taken into consideration in all of our performance analyses. Overall, our results demonstrate that the BGC enabled subgraph pruning achieves speedups of up to 1.6× compared to the unpruned case in Amazon2M and Ogbl-Citation2, and average speedup of 1.25× on Flickr and Yelp across GCN, SAGE, and GAT. BGC-enabled subgraph pruning leads to performance speedup during GNN training as the number of input subgraphs processed during training

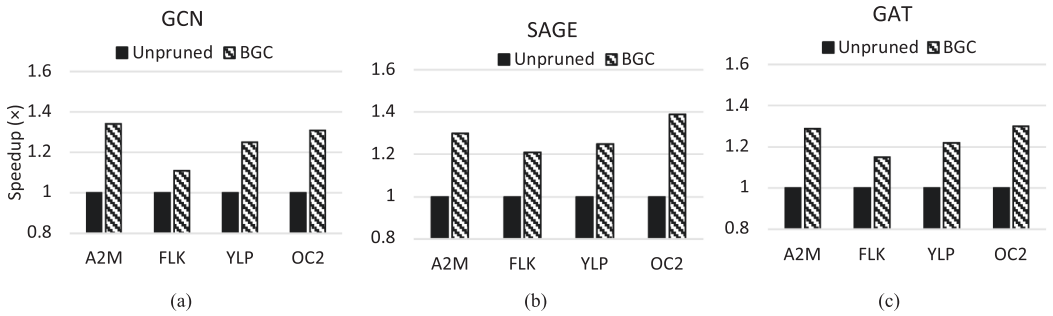

Fig. 6. (a)-(c) GNN training performance speedup of unpruned GNN model for Amazon2M (A2M), Flickr (FLK), Yelp (YLP) and Ogbl-citation2 (OC2) compared to BGC enabled subgraph pruning applied at epoch 5 for GCN, SAGE and GAT models.

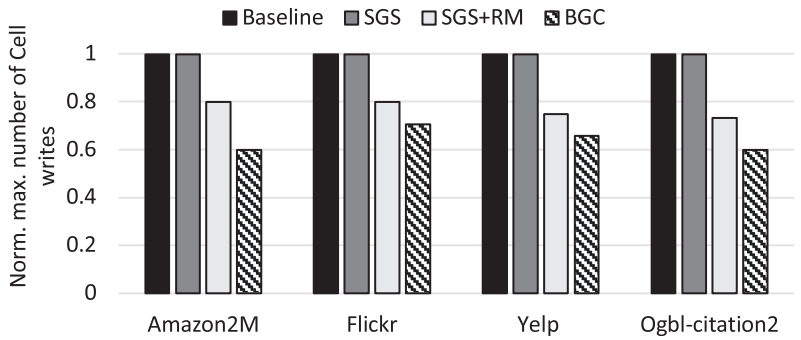

Fig. 7. Total worst-case number of writes for Amazon2M, Flickr, Yelp, and Ogbl-citation2 normalized w.r.t. to their respective enduracnce-unware baseline.

reduces. As a result, this reduces the overall pipeline depth, reduces the end-to-end latency, and extends the lifespan of ReRAM-crossbars due to fewer cell write operations. Moreover, the power and performance can be improved further by synergistically combining the proposed BGC-enabled data pruning method with other techniques such as weight pruning and/or quantization [33].

4.4 Reliability Improvement due to Subgraph Pruning

In this sub-section, we discuss the benefits in terms of the possible number of GNN training tasks that can be executed on ReRAM crossbars enabled by the BGC-enabled subgraph pruning method. The BGC-enabled data pruning technique reduces the number of ReRAM cell writes during training. Consequently, this extends the lifespan of the ReRAM crossbar, as more training instances can be performed on it before it reaches its endurance limit. Hence, subgraph pruning enables reliable GNN training on ReRAM-based PIM architectures especially for workloads with large graphs datasets.

To adequately evaluate the improvement in reliability due to the BGC-enabled pruning, we first quantify the worst-case number of write operations on a ReRAM cell during a GNN training instance. We then compare this to the number of ReRAM crossbar cell writes for a GNN training instance with the entire graph dataset (all subgraphs). For our analysis, we look at the ReRAM crossbar write distributions, and consider the maximum crossbar cell write count. This gives the worst-case number of writes for the ReRAM crossbar. The ReRAM cell with the maximum or worst-case number of writes will reach the endurance limit and fail first. The failure of a single

Table 6. Increase in Number of GNN Training Instances for the ReRAM-based PIM Architecture

	Amazon2M	Flickr	Yelp	Ogbl-citation2
SGS+RM	1.25×	1.20×	1.33×	1.36×
BGC	1.64×	1.31×	1.48×	1.62×

ReRAM cell results in the outcome that the whole crossbar array produces incorrect results. Here, we consider an endurance limit of 10^6 write operations, as this is in line with fabricated ReRAM devices [9, 43].

As discussed in Section 2 above, a recent endurance aware training technique known as **structured gradient sparsification (SGS)** has been proposed for ReRAM-based PIM architectures. SGS performs gradient accumulation, and prunes gradients based on their magnitude, or in a stochastic fashion thereby reducing the frequency of weight updates [23]. In addition, SGS also takes into account the shape of the crossbar array and incorporates **row remapping (RM)** to reduce the number of writes to the ReRAM crossbar and balance out the write across crossbars. Consequently, this improves the reliability of ReRAM-based PIM architectures. Figure 7 shows the worst-case total number of write operations on to a ReRAM cell during a GCN training instance for all the four datasets: Amazon2M Reddit, Yelp, and Ogbl-citation2. Here, the maximum (or worst-case) number of writes per ReRAM-cell for each endurance-aware training method (SGS, SGS+RM and BGC) is normalized with respect to the endurance-unaware training baseline. We train each GNN model (GCN, SAGE and GAT) for the same number of training iterations (200 epochs), while keeping the number of subgraphs consistent across all the GNN models. As shown in Figure 7, the Ogbl-citation2 dataset has the highest number of write operations (over 200K writes in the baseline case) for a GNN training instance. This is attributed to the large size of the graph and high number of input training subgraphs. Meanwhile, Flickr has the lowest worst-case number of write operations on to the ReRAM crossbar cells due to its small size and small number of subgraphs. In this work, we compare our proposed BGC approach with SGS in terms of the total number of write operations, and the improvement in possible number of training instances as shown in Figure 7. In addition to SGS, we incorporate a remapping-based (RM) technique that remaps weights, activations, and adjacency matrices to even-out the write operations across crossbars [23]. As shown in Figure 7, the standalone SGS technique has the same worst-case number of writes as the baseline case without any endurance-aware training. This happens as despite its advantages, SGS is not well suited for GNN training, as it focuses solely on reducing the number of write operations due to weight updates and does not account for ReRAM cell writes due to activations and graph adjacency matrix storage. As a result, this only improves the reliability of ReRAM cells storing the weights. Meanwhile, cells storing activations and the adjacency matrices are not impacted by SGS. Hence, these cells will fail early due to repeated writes, thereby jeopardizing the whole GNN training process. Moreover, SGS results in performance degradation due to frequent row remapping. This is due to the additional timing overhead incurred by the top-k gradient selection that must be performed for pruning and row remapping at run-time. However, when remapping is incorporated with SGS (SGS+RM) to balance out the number of writes across crossbars, it reduces the worst-case number of writes compared to the baseline. In Table 6, we show the improvement in the number of possible training instances for SGS+RM, and the proposed BGC enabled GNN training with respect to the baseline method. The SGS+RM method can enable up to a ~1.4× increase in the number of training instances before failing, as shown in Table 6. However, the SGS+RM reliability improvement method incurs significant on-chip area and energy overhead for remapping workloads at runtime to achieve even write distribution, and achieve lower number of writes

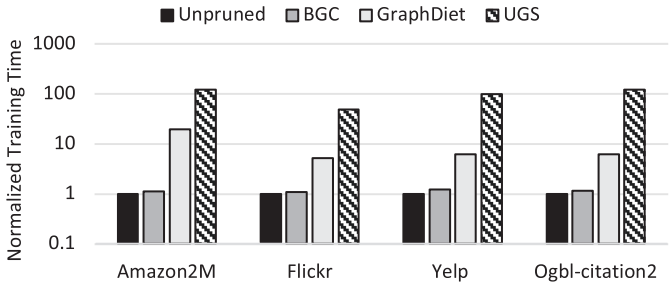

Fig. 8. Normalized training time required using the BGC, GraphDiet, and UGS data pruning method for Amazon2M, Flickr, Yelp, and Ogbl-citation2 datasets. Each method is normalized with respect to the time required for training the GNN on the unpruned dataset.

[23, 24, 44]. Unlike this method, the proposed BGC-enabled pruning framework is a simple yet effective method that effectively reduces the worst-case total number of write operations to the ReRAM, thereby achieving up to a 1.6× improvement in the number of possible training instances as shown in Table 6. Hence, this leads to more reliable training on ReRAM-enabled PIM accelerators without incurring any additional hardware area and energy cost.

4.5 Training Cost, Reliability, and Performance Trade-offs

Training Cost: As discussed in Section 2 on related prior work, data pruning is a well-known method to reduce the amount of computation performed during training of neural networks [31, 45]. As discussed in Section 1, we specifically define the training cost as the amount of training time required to find the maximally pruned dataset that can be trained with the GNN without any accuracy loss. However, most existing data pruning techniques usually incur high training cost, as they require multiple rounds of training to maximally prune the dataset [3]. For example, as discussed in Section 2, existing LTH-based pruning methods require multiple training rounds/instances to prune input graphs such as the UGS method. Also, a **score-based data pruning (SDP)** approach that relies on few initial training epochs to prune subgraphs for GNN training known as 'GraphDiet' was recently proposed in [33]. This method also prunes subgraphs during GNN training. However, it requires initial training on the new datasets (Amazon2M, Flickr, Yelp, and Ogbl-citation2) to identify important and unimportant subgraphs based on a score. Moreover, GraphDiet chooses the pruning percentage (p) by repeating the training process to find the appropriate pruning percentage for a new dataset. As a result, this makes it difficult to transfer the GraphDiet mechanism to unseen or new datasets without prior training with the new graph dataset. Unlike GraphDiet, BGC offers a one-time predictive pruning approach that is transferrable to new or unseen graph datasets. The BGC relies on a one-time supervised learning approach to learn the patterns of subgraph structures in terms of node features/labels and their connectivity for subgraphs that can pruned, and subgraphs that are important for subsequent training. Hence, this generalized knowledge can be transferred to new or unseen datasets as demonstrated by our strong experimental results across diverse unseen graph datasets.

For our analysis, we compare the overall cost in terms of training time required for existing data pruning approaches with our proposed BGC-enabled subgraph pruning. Figure 8 shows a comparison between UGS, GraphDiet, and the proposed BGC-based graph pruning methods in terms of the total training time cost. Here, all three methods considered (UGS, GraphDiet, and BGC) are normalized with respect to the time required for GNN training on the unpruned datasets (Amazon2M Flickr, Yelp, and Ogbl-citation2). We incorporate the training time required for the

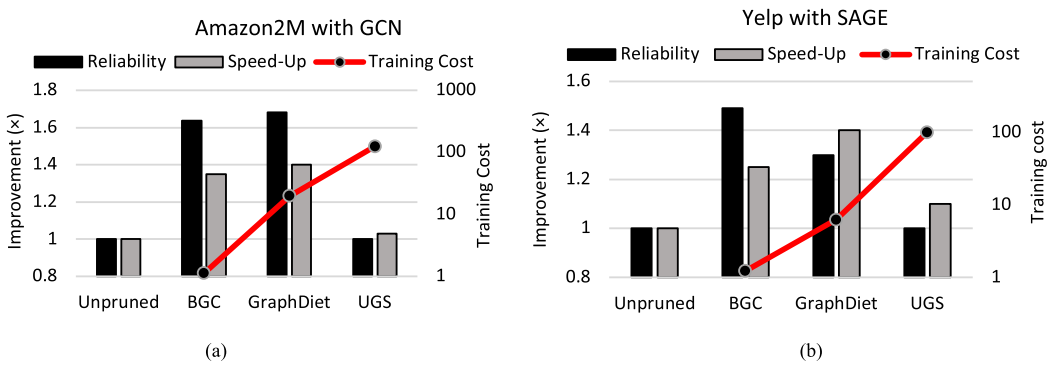

Fig. 9. (a) and (b): Improvement in reliability and performance speedup vs. training cost for BGC, GraphDiet, and UGS graph pruning methods used on the Amazon2M and Yelp datasets. Each method is normalized with respect to the unpruned version of the dataset.

MLP-based BGC classifier. Given that the BGC is an MLP based model with only two feed-forward layers and a ReLU layer, its training time is very small compared to the training of an entire GNN. Moreover, existing methods require some form of training on the new dataset before it can be pruned. The proposed BGC model is trained only once with selected known datasets (PPI and Reddit). As a result, the one-time training and transferability of the BGC-enabled pruning framework to new or unseen graph dataset and amortizes its training cost over multiple GNN training instances with new datasets, as no extra training is required for a new or unseen graph dataset. As shown in Figure 8, the cost of training required to prune new datasets (Amazon2M, Flickr Yelp, and Ogbl-citation2) is quantified. The UGS method requires up to 100× more training time compared to a training scenario without any data pruning (unpruned) for Amazon2M as an example. We observe a similar trend across other datasets and models. Meanwhile, GraphDiet requires about 8.5× more training time on average for all datasets as shown in Figure 8. This additional timing overhead is required for the initial training epochs on the new dataset before it can be pruned without any accuracy loss. For a fair comparison of UGS and GraphDiet with our proposed BGC-enabled pruning approach, we ensure that UGS and GraphDiet achieves the same percentage of pruning (p) with BGC in each dataset.

Reliability, Speedup, and Training Cost Trade-off: We also perform a trade-off analysis of the reliability, speedup, and training cost for the three data pruning methods (BGC, GraphDiet, and UGS). In Figure 9(a) and Figure 9(b), we show the key figure of merits (reliability, performance speedup, and training cost) together, and show the trade-off for the Amazon2M and Yelp datasets trained with the GCN and SAGE GNN models, respectively. Note that we see a similar trend in our trade-off analysis with the Flickr and Ogbl-citation2 datasets, and the GAT model. Here, the reliability, and performance speedup, for BGC, UGS, and GraphDiet, are normalized with respect to the unpruned case for each dataset under consideration. As shown in Figure 9(a) and Figure 9(b), the normalized training cost increases exponentially for UGS, while there is a low improvement in reliability and performance speedup of less than 1.2×. This is because UGS only prunes graph edges on the entire monolithic graph. Hence, when the pruned monolithic graph is partitioned into subgraphs, each individual subgraph is sparse, but the total number of subgraphs (K) still remains the same. As a result, this still leads to high number of ReRAM cell writes and thus low reliability as the unpruned case. The slight improvement in performance speed-up observed in UGS is attributed to a reduced number of MVM operations performed due to model and graph pruning. However, this has little to no impact on the reliability of the ReRAM-based PIM accelerator and

GNN training speedup. We observe a higher improvement in performance speedup and reliability for the BGC and GraphDiet as shown in Figure 9(a) and Figure 9(b). For the Amazon2M dataset in Figure 9(a), GraphDiet achieves slightly improved performance in terms of speedup and reliability over BGC, while it is slightly lower for the Yelp dataset. However, Graph Diet has a training cost that is 10× more than the BGC in Amazon2M, and 8.8× more than the BGC in Yelp. This is because the GraphDiet pruning percentage ($p = 48.0\%$) for the Amazon2M dataset with GCN is 5% higher than BGC's pruning percentage ($p = 43.1\%$). Meanwhile, GraphDiet technique achieves a slightly lower pruning percentage of 28%, while BGC prunes up to 30.5% for the Yelp dataset with SAGE. Nonetheless, the GraphDiet method still incurs significantly more training time cost than BGC in both cases (Figure 9(a) and Figure 9(b)), as the first GNN needs to be trained on the unseen dataset before it can be pruned. Unlike UGS and GraphDiet, our proposed BGC-enabled pruning only relies on a light-weight model trained once on known datasets and can be transferred to new datasets without requiring prior GNN training. In addition, the BGC incurs the least training time cost, as it relies on simple MLP network trained only once, that can be transferred to new datasets. Overall, the BGC method achieves the best performance speedup, reliability, and training time cost trade-off when compared to UGS and GraphDiet.

4.6 Performance Analysis of BGC-enabled GNN Training on ReRAM-based PIM Architecture

In this subsection, we present a thorough performance evaluation of the ReRAM-based 3D PIM architecture outlined in Table 4 by incorporating the proposed BGC framework. We compare the performance of the PIM architecture with BGC with respect to other existing PIM-enabled many-core architectures for GNN training. For this comparison, we consider two other ReRAM-based architectures DARe and ReMaGN, which were designed for GNN training [6, 19]. The ReMaGN architecture uses low-precision with stochastic rounding to improve the performance of GNN training on ReRAM-based architectures. Meanwhile, the DARe framework, leverages dropout and DropEdge to improve the computation and communication efficiency of ReMaGN. Hence, DARe achieves better performance and energy-efficiency compared to ReMaGN. We consider overall speedup, **energy-delay-product (EDP)**, and resource utilization as the relevant metrics in this comparative performance evaluation. Here, we define the resource utilization as the amount of hardware resources (number of ReRAM crossbars) required for training the same GNN model using the three frameworks under consideration (ReMaGN, DARe, and BGC). For this comparative performance evaluation, we consider three datasets, viz., PPI, Reddit and Amazon2M for the sake of brevity. However, we observe a similar trend in speedup and EDP improvement across the other datasets (Flickr, Yelp and Ogbl-citation2). Figure 10(a) and Figure 10(b) show the speedup, and energy delay product (EDP) respectively of the proposed BGC framework compared to DARe, and ReMaGN and a conventional GPU-baseline (Nvidia V100 in this case). It is evident that all the ReRAM-based architectures outperform the GPU as shown in Figure 10(a) and Figure 10(b). However, among all the ReRAM-based architectures BGC achieves the best performance. Overall, our proposed BGC framework achieves on an average speedup improvement over ReMaGN and DARe by 1.6× and 1.3×, respectively. Similarly, we also observe on an average EDP improvement in BGC of 5.2× and 2.5× with respect to ReMaGN and DARe, respectively. In Figure 10(c), we compare the ReRAM utilization (i.e., number of ReRAM crossbars required for training) for the BGC framework normalized with respect to ReMaGN and DARe. It should be noted that DARe utilizes a software technique (viz., Dropout and DropEdge-based regularization) to improve the performance of ReMaGN. Hence, DARe and ReMaGN have exactly similar hardware utilization. BGC utilizes 35% less ReRAM crossbars on an average compared to DARe and ReMaGN as it prunes subgraphs, thereby reducing the storage requirement for graph adjacency matrices.

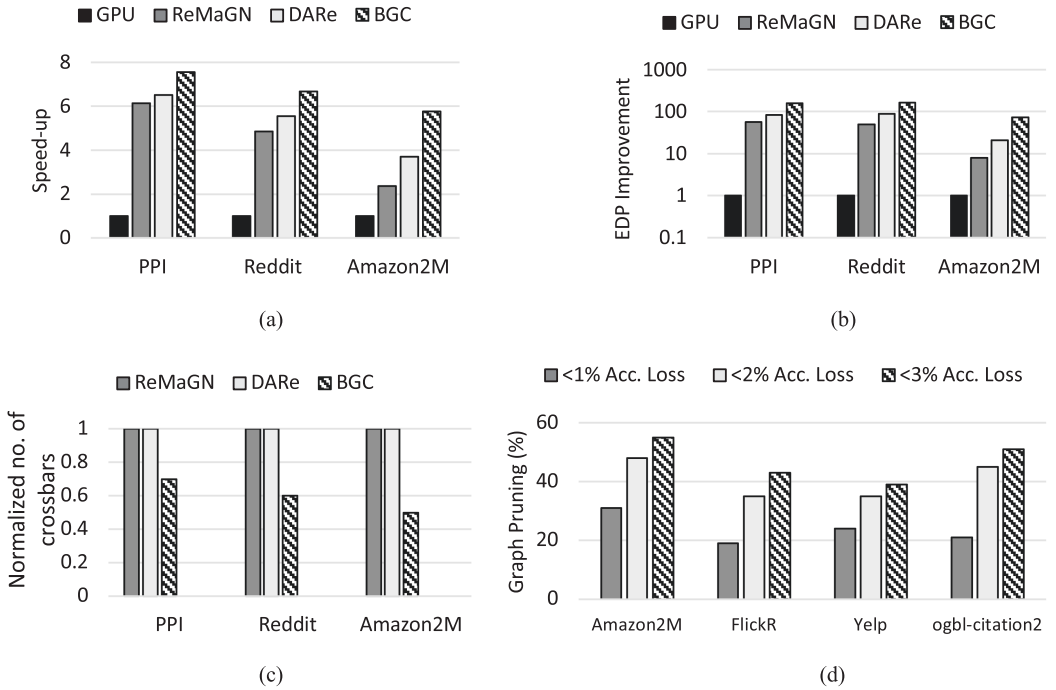

Fig. 10. (a) Performance Speedup, and (b) EDP improvement of BGC compared to ReMaGN, and DARe, all normalized with respect to GPU. (c) ReRAM utilization (i.e., number of ReRAM crossbars required for training) for the BGC framework normalized with respect to ReMaGN and DARe. (d) Allowable accuracy loss threshold vs graph pruning percentage.

Graph pruning vs accuracy-loss trade-off: We have also performed ablation experiments to determine the corresponding graph pruning percentage of BGC for various accuracy loss thresholds of 1%, 2% and 3%. To perform this ablation study, we set these accuracy thresholds in the process of generating the supervised binary classification data (i.e., Algorithm 1). The maximum value of k for each accuracy threshold is chosen such that removing k% of subgraphs does not lead to an accuracy loss beyond the set threshold. Figure 10(d) shows the accuracy loss and the corresponding graph pruning percentage of the Amazon2M, Flickr, Yelp, and Ogbl-citation2 datasets using different accuracy thresholds. As shown in Figure 10(d), for an accuracy loss of less than 1% (<1%), 2% (<2%), and 3% (<3%), we obtain a corresponding average graph pruning percentage of 23.75%, 40.75% and 47% respectively across all four datasets. As observed, the amount of pruning that can be achieved in each dataset is directly proportional to the corresponding accuracy loss, i.e., more pruning leads to more accuracy loss. Hence, the achievable improvement in performance and reliability is limited due to significant accuracy loss penalty. As a result, there exists a trade-off between the allowable accuracy loss and percentage of graph pruning both of which have an impact on the overall performance, and reliability of the ReRAM-based PIM architecture. Pruning more subgraphs is desirable from a design perspective, as this leads to significant improvement in speedup and reliability (due to fewer write operations). However, certain GNN applications have stringent accuracy requirements which limit the amount of pruning that the BGC can achieve as shown in Figure 10(d). Therefore, depending on the GNN application under consideration, the BGC can achieve different levels of speed-up and improvement in reliability. The user can choose the BGC setting based on the desired trade-off between performance, reliability and accuracy. Here,

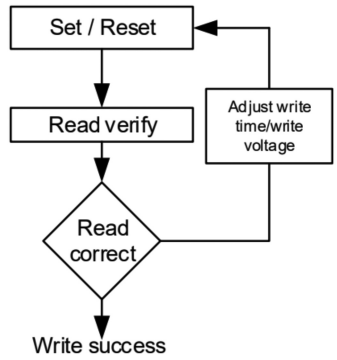

Fig. 11. Overview of the write-verify scheme [48].

Table 7. Max. Number of Writes, and Increase in Number of GNN Training Instances with and without Write-Verify (WV)

Dataset	No. of writes without WV	No. of writes with WV	No. of possible training instances without WV	No. of possible training instances with WV
Amazon2M	360K	931K	~290 times	~107 times
Amazon2M+BGC	203K	542K	~500 times	~184 times

we do not show graph pruning percentages for accuracy loss beyond 3% (in Figure 10(d)), as such accuracy loss in many real world GNN applications may not be permissible.

4.7 Endurance of BGC-enabled GNN Training Accelerator

Prior work has reported ReRAM endurance that ranges from 10^6 to 10^{12} writes [11, 21]. However, the number of times a GNN model can be trained before the ReRAM devices reach the endurance limit depends on the graph dataset under consideration. A single training epoch requires a certain number of ReRAM crossbar writes which is directly proportional to the number of input subgraphs (K), and one training process (or instance) is typically made-up of multiple training epochs (E). Hence, the total number of write operations during the training process is proportional to $E \times K$. For example, with an average case endurance of 10^8 writes, a GNN model can be trained with the Amazon2M dataset (consisting of 1,000 subgraphs) for approximately five hundred (~500) times using the BGC framework before the ReRAM cells begin to fail. However, considering the write variation problems, i.e., a single write operation makes it difficult to tune the ReRAM device to the target resistance value, the available training times of the ReRAM-based architecture may be less than 500 times. Hence, we need to consider the write variation problem in ReRAM, i.e., a single write operation makes it difficult to tune the ReRAM device to the target resistance value. Prior work has proposed the write verify technique to address the write variation problems of ReRAM devices and ensure robust training on ReRAM-based platforms [46, 47]. Figure 11 shows an overview of the write verify technique [48]. However, incorporating the write-verify technique during GNN training increases the number of crossbar cell write operations during each programming phase, due to repetitive application of a write voltage for the SET/RESET operation. Hence, this limits the number of training instances that can be performed before the device reaches its endurance limit.

In this work, we have adopted a simple write-verify technique proposed in [48] into our BGC-enabled GNN training framework. In Table 7, we show the total number of writes, and the number

of training instances that can be performed with and without the incorporation of a write-verify scheme. Here, we consider an endurance limit of 10^8, for both the unpruned and BGC-pruned Amazon2M (Amazon2M+BGC) dataset trained with the GCN model as an example. However, note that the write-verify scheme is model- and dataset-agnostic, hence a similar trend is observed across other datasets (Flickr, Yelp and ogbl-citation2) and models (SAGE and GAT). As shown in Table 7, we can observe that with the write-verify scheme incorporated during GNN training, the total number of writes during training increases by over ∼2.5× on average. Hence, this limits the number of possible training instances that can be executed on the ReRAM-based PIM platform before the ReRAM cell with the worst-case number of writes reaches its endurance limit of 10^8 writes and fails first. As shown in Table 7, the Amazon2M dataset can be trained for about 500 times without write-verify, and 184 times with the write-verify technique. Also, we observe that the BGC enabled GNN training on the ReRAM-based PIM architecture consistently achieves an improvement of over ∼1.6× in the number of training instances compared to its unpruned counterpart with and without the write-verify scheme. Hence, this demonstrates that our proposed BGC method is compatible with the write-verify technique, and still enhances the possible GNN training instances compared to the unpruned version. In summary, the BGC enabled data pruning approach is orthogonal to the write-verify scheme, as well as other write optimization algorithms [11, 49], hence they can be incorporated with BGC to improve the available GNN training instances for ReRAM-based PIM architectures.

5 CONCLUSION

ReRAM-based non-volatile memories enable the design of high-performance and energy-efficient architectures for accelerating GNN training [19, 50]. However, the limited write endurance of ReRAM crossbar cells decreases their reliability for GNN training. Moreover, the end-to-end execution time for GNN training, and the number of write operations to crossbar cells in ReRAM-based PIM architectures is determined by the number of subgraphs (size of input data graph workload). We have presented a novel subgraph pruning framework based on Binary Graph Classification (BGC) to reduce the number of input subgraphs without sacrificing the GNN model accuracy to improve the performance and reliability of GNN training on ReRAM-based architectures. The BGC model leverages a pre-trained MLP Neural Network to perform pruning predictions on subgraphs very early during the GNN training process to improve the performance and reliability of ReRAM-based PIM architectures. The BGC-enabled pruning framework improves the reliability of ReRAM-based architectures and accelerates GNN training by up to 1.6× compared to its unpruned counterpart without any significant model accuracy on ReRAM-based PIM architectures. Overall, the proposed method reduces the overall training cost by up to 100× compared to state-of-the-art graph pruning techniques.

REFERENCES

[1] R. Ying, R. He, K. Chen, P. Eksombatchai, W. L. Hamilton, and J. Leskovec. 2018. Graph convolutional neural networks for web-scale recommender systems. In *ACM SIGKDD International Conference on Knowledge Discovery & Data Mining*, London.

[2] T. Kipf and M. Welling. 2017. Semi-supervised classification with graph convolutional networks. In *International Conference on Learning Representations (ICLR)*.

[3] A. Liu et al. 2023. Comprehensive graph gradual pruning for sparse training in graph neural networks. *IEEE Transactions on Neural Networks and Learning Systems* (2023).

[4] W. Fan et al. 2019. Graph neural networks for social recommendation. In *The World Wide Web Conference*, San Francisco, CA.

[5] D. Xu et al. 2020. Edge Intelligence: Architectures, Challenges, and Applications. http://arxiv.org/abs/2003.12172

[6] A. I. Arka, B. K. Joardar, J. R. Doppa, P. P. Pande, and K. Chakrabarty. 2021. DARe: DropLayer-aware manycore ReRAM architecture for training graph neural networks. In *International Conference on Computer-Aided Design (ICCAD)*.

[7] A. Shafiee et al. 2016. ISAAC: A convolutional neural network accelerator with in-situ analog arithmetic in crossbars. In *International Symposium on Computer Architecture (ISCA)*.

[8] P. Chi et al. 2016. PRIME: A novel processing-in-memory architecture for neural network computation in ReRAM-based main memory. In *International Symposium on Computer Architecture (ISCA)*.

[9] K. Roy, I. Chakraborty, M. Ali, A. Ankit, and A. Agrawal. 2020. In-memory computing in emerging memory technologies for machine learning: An overview. In *IEEE Design Automation Conference (DAC)*.

[10] S. Resch et al. 2023. On endurance of processing in (nonvolatile) memory. In *Proceedings - International Symposium on Computer Architecture (ISCA)*.

[11] W. Wen, Y. Zhang, and J. Yang. 2019. ReNEW: Enhancing lifetime for ReRAM crossbar based neural network accelerators. In *IEEE International Conference on Computer Design (ICCD)*.

[12] T. Chen et al. 2021. A unified lottery ticket hypothesis for graph neural networks. In *International Conference on Machine Learning (ICML)*.

[13] H. You, Z. Lu, Z. Zhou, Y. Fu, and Y. Lin. 2022. Early-bird GCNs: Graph-network co-optimization towards more efficient GCN training and inference via drawing early-bird lottery tickets. In *AAAI Conference on Artificial Intelligence*.

[14] J. Frankle and M. Carbin. 2019. The lottery ticket hypothesis: Finding sparse, trainable neural networks. In *International Conference on Learning Representations (ICLR)*.

[15] C. Ogbogu et al. 2022. Accelerating large-scale graph neural network training on crossbar diet. In *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*.

[16] P. Veličković et al. 2017. Graph attention networks. arXiv preprint arXiv:1710.10903.

[17] W. Hamilton, Z. Ying, and J. Leskovec. 2017. Inductive representation learning on large graphs. In *Advances in Neural Information Processing Systems* 30.

[18] A. I. Arka, B. K. Joardar, J. R. Doppa, P. P. Pande, and K. Chakrabarty. 2021. ReGraphX: NoC-enabled 3D heterogeneous ReRAM architecture for training graph neural networks. In *Design, Automation & Test in Europe Conference & Exhibition (DATE)*.

[19] A. I. Arka, B. K. Joardar, J. R. Doppa, P. P. Pande, and K. Chakrabarty. 2021. Performance and accuracy tradeoffs for training graph neural networks on ReRAM-based architectures. In *IEEE Transactions on Very Large Scale Integration (VLSI) Systems* 29, 10 (2021), 1743-1756.

[20] L. Chen et al. 2017. Accelerator-friendly neural-network training: Learning variations and defects in RRAM crossbar. In *Design, Automation & Test in Europe Conference & Exhibition (DATE)*.

[21] W. Wen, Y. Zhang, and J. Yang. 2018. Wear leveling for crossbar resistive memory. In *ACM /IEEE Design Automation Conference (DAC)*.

[22] X. Yang et al. 2022. ESSENCE: Exploiting structured stochastic gradient pruning for endurance-aware ReRAM-based in-memory training systems. In *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*.

[23] Y. Cai et al. 2018. Long live TIME: Improving lifetime for training-in-memory engines by structured gradient sparsification. In *Proceedings - Design Automation Conference (DAC)*.

[24] L. Xia et al. 2017. Fault-tolerant training enabled by on-line fault detection for RRAM-based neural computing systems. In *IEEE Design Automation Conference (DAC)*.

[25] V. Sadhanala, Y. X. Wang, and R. Tibshirani. 2016. Graph sparsification approaches for Laplacian smoothing. *Artificial Intelligence and Statistics*. PMLR.

[26] D. Calandriello et al. 2018. Improved large-scale graph learning through ridge spectral sparsification. *International Conference on Machine Learning*. PMLR.

[27] Z. Cheng et al. 2020. Robust graph representation learning via neural sparsification. *International Conference on Machine Learning*. PMLR.

[28] W. Chiang et al. 2019. Cluster-GCN: An efficient algorithm for training deep and large graph convolutional networks. In *Proceedings of the ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*.

[29] W. Hu, M. Fey, M. Zitnik, Y. Dong, H. Ren, B. Liu, M. Catasta, and J. Leskovec. 2020. Open graph benchmark: Datasets for machine learning on graphs. In *34th Conference on Neural Information Processing Systems (NeurIPS)*, Vancouver, Canada.

[30] K. Xu et al. 2018. How powerful are graph neural networks? arXiv preprint arXiv:1810.00826.

[31] X. Peng et al. 2020. DNN+NeuroSim V2.0: An end-to-end benchmarking framework for compute-in-memory accelerators for on-chip training. arXiv:2003.06471.

[32] G. Karypis and V. Kumar. 1998. A fast and high quality multilevel scheme for partitioning irregular graphs. *SIAM Journal on Scientific Computing*.

[33] C. O. Ogbogu et al. 2023. Accelerating graph neural network training on ReRAM-based PIM architectures via graph and model pruning. In *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems* 42 (2023), 2703–2716.

[34] B. K. Joardar et al. 2021. Learning to train CNNs on faulty ReRAM-based manycore accelerators. *ACM Transactions on Embedded Computing Systems* (2021), 1–23.
[35] M. Paul, S. Ganguli, and G. K. Dziugaite. 2021. Deep learning on a data diet: Finding important examples early in training. In *Advances in Neural Information Processing Systems* 34 (NeurIPS 2021).
[36] T. H. Phan and K. Yamamoto. 2020. Resolving class imbalance in object detection with weighted cross entropy losses. arXiv preprint arXiv:2006.01413.
[37] C. Zhang, F. Wei, Q. Liu, Z. G. Tang, and Z. Li, 2017. Graph edge partitioning via neighborhood heuristic. In *KDD*.
[38] K. Cao et. al. 2023. Learning large graph property prediction via graph segment training. [Online]. Available: https://arxiv.org/abs/2305.12322
[39] B. K. Joardar et al. 2020. AccuReD: High accuracy training of CNNs on ReRAM/GPU heterogeneous 3D architecture. *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*.
[40] S. Gupta, A. Agrawal, P. Narayanan, and K. Gopalakrishnan. 2015. Deep learning with limited numerical precision. In *International Conference on Machine Learning (ICML)*.
[41] N. Agarwal, T. Krishna, L. Peh, and N. Jha. 2009. GARNET: A detailed on-chip network model inside a full-system simulator. In *International Symposium on Performance Analysis of Systems and Software (ISPASS)*.
[42] T.-T. Wong. 2015. Performance evaluation of classification algorithms by k-fold and leave-one-out cross validation. *Pattern Recognition* (2015), 2839–2846.
[43] L. Yavits et al. 2020. WoLFRaM: Enhancing wear-leveling and fault tolerance in resistive memories using programmable address decoders. In *IEEE 38th International Conference on Computer Design (ICCD)*.
[44] Y. Xiaoxuan et al. 2022. ESSENCE: Exploiting structured stochastic gradient pruning for endurance-aware ReRAM-based in-memory training systems. *IEEE TCAD*.
[45] M. Hwang, Y. Jeong, and W. Sung. 2020. Data distribution search to select core-set for machine learning. In *International Conference on Smart Media and Applications*.
[46] C. Xu, D. Niu, N. Muralimanohar, N. P. Jouppi, and Y. Xie. 2013. Understanding the trade-offs in multi-level cell ReRAM memory design. In *50th ACM/EDAC/IEEE Design Automation Conference (DAC)*, Austin, TX.
[47] W. Shim, S. Jae-sun, and S. Yu. 2020. Two-step write–verify scheme and impact of the read noise in multilevel RRAM-based inference engine. *Semiconductor Science and Technology* (2020) 35.
[48] Y. Zhang, L. Zhu, L. Zhang, and Z. Wang. 2019. A write-verification method for non-volatile memory. In *International Conference on IC Design and Technology (ICICDT)*.
[49] J. Zhang et al. 2023. Realizing extreme endurance through fault-aware wear leveling and improved tolerance. In *2023 IEEE International Symposium on High-Performance Computer Architecture (HPCA)*.
[50] A. Auten, M. Tomei, and R. Kumar. 2020. Hardware acceleration of graph neural networks. In *IEEE Design & Automation Conference (DAC)*.

Received 2 November 2023; revised 13 March 2024; accepted 23 March 2024