LiGNN: Accelerating GNN Training through Locality-aware Dropout

Gongjian Sun[*†], Mingyu Yan[*†], Dengke Han[*†], Runzhen Xue[*†], Xiaochun Ye[*†], Dongrui Fan[*†]
[*]State Key Lab of Processors, Institute of Computing Technology, CAS
[†]University of Chinese Academy of Sciences

***Abstract*—Graph Neural Networks (GNNs) have demonstrated significant success in graph learning and are widely adopted across various critical domains. However, the irregular connectivity between vertices leads to inefficient neighbor aggregation, resulting in substantial irregular and coarse-grained DRAM accesses. This lack of data locality presents significant challenges for execution platforms, ultimately degrading performance. While previous accelerator designs have leveraged on-chip memory and data access scheduling strategies to address this issue, they still inevitably access features at irregular addresses from DRAM.**

In this work, we propose LiGNN, a hardware-based solution that enhances locality and applies dropout to aggregation to accelerate GNN training. Unlike algorithmic dropout approaches that primarily focus on improving accuracy and neglects hardware costs, LiGNN is specifically designed to drop nodes' features with data locality awareness, directly targeting the reduction of irregular DRAM accesses, meanwhile maintaining accuracy. LiGNN introduces locality-aware ordering and a DRAM row integrity policy, enabling configurable burst and row-granularity dropout at the DRAM level. This approach improves data locality and ensures more efficient DRAM access. Compared to state-of-the-art methods, under classic 0.5 droprate, LiGNN achieves a 1.6∼2.2× speedup, reduces DRAM accesses by 44∼50% and DRAM row activation by 41∼82%, all without losing accuracy.

***Index Terms*—Graph neural network, data locality, dropout**

I. Introduction

Graph Neural Networks (GNNs) are a class of neural networks designed to operate on graph-structured data, effectively capturing relationships between entities [27]. GNNs excel at tasks where data is naturally represented as nodes and edges, making them powerful for modeling complex systems. GNNs have gained wide application in many critical fields such as electronic design automation (EDA) [16], knowledge inference [22], recommendation system [35], visual reasoning [6], traffic prediction [17], and so on [13], [27].

A typical GNN model consists of multiple layers, each of which contains two main phases: aggregation and combination. In the aggregation phase, each node collects messages from its neighbors to generate an intermediate result, using an order-independent operator like element-wise sum or mean. This iteration process follows the sparse and irregular graph structure, where massive memory accesses with poor locality lead to significant performance degradation. While in the combination phase, the intermediate result and features of the node are passed to a feedforward or dense layer, which shows regular execution pattern and high data reusability. In all, the aggregation phase is memory-bound and time-consuming, thus becomes the main bottleneck of GNN models [32], [34].

Robustness, or tolerance of data corruption, is a noteworthy characteristic of GNNs, potentially stemming from their intrinsic capability to capture structural dependencies. Several studies have leveraged dropout method in GNN aggregation to enhance accuracy [7], [11], [14], [19], all proved effective.

Therefore, not all accesses to node features are essential, presenting an opportunity to selectively drop certain irregular accesses to improve data locality and achieve speedup. However, algorithmic dropout methods inherently lack awareness of the DRAM working mechanism, making them incapable of making DRAM-friendly dropout decisions. Accordingly, we propose LiGNN, a data locality extraction and enhancement hardware solution customized for dropout in GNN training. LiGNN seats between DRAM and GNN training accelerator, intervening DRAM read accesses between them. We implement hardware dropout by conditionally dropping DRAM read access to node features in the aggregation phase, considering locality. We primarily focus on DRAM, because GNNs have ultra low locality in the aggregation phase and almost all read requests are ultimately served by DRAM. It is worth noting that our work is not a straight forward hardware implementation for previous algorithmic dropout efforts, but a novel design that enhance data locality inside DRAM following GNN robust nature. To the best of our knowledge, no existing work has implemented dropout for GNNs as hardware-based filter seizing the opportunity provided by GNN robustness.

The key contributions of our work are as follows:

- We identify an unexplored approach to enhance GNN aggregation locality and gain speedup by dropping DRAM access, in accordance with the robust nature of GNNs.
- We introduce LiGNN, a hardware-based solution designed to enhance locality for DRAM accesses, specifically optimized to accelerate neighbor aggregation in GNN training.
- We propose a locality-aware ordering mechanism to group desired accesses into actual accesses based on their locations in DRAM, enabling burst dropout and reducing DRAM accesses. Additionally, we implement a DRAM row integrity policy with customized criteria for row filtering to balance the efficiency of dropping and the utilization of open rows, streamlining DRAM row activation.

This work was supported by National Key Research and Development Program (Grant No. 2022YFB4501400), the National Natural Science Foundation of China (Grant No. 62202451 and 6230247), CAS Project for Young Scientists in Basic Research (Grant No. YSBR-029), and CAS Project for Youth Innovation Promotion Association. Mingyu Yan is the corresponding author. (e-mail: yanmingyu@ict.ac.cn).

TABLE I
NOTATIONS USED IN THIS PAPER.

Notation	Explanation
$G = (V, E)$	graph G with vertices V and edges E
v or v_i	vertex v (with index i)
(i, j) or $e^k_{i,j}$	(k-th feature of) edge from vertex i to j
d_v, d^+_v, d^-_v	common, out, in degree of vertex v
N_v, N^+_v, N^-_v	common, out, in neighbor set of vertex v
x^k_v	input feature of vertex v at k-th layer
h^k_v	intermediate result of vertex v at k-th layer

- We implement LiGNN in both RTL and cycle-accurate simulator, based on state-of-the-art (SOTA) GNN training accelerator GCNTrain [15] and evaluate it on several models, datasets and DRAM standards. The results show that under classic 0.5 droprate, LiGNN achieves 1.6∼2.2x speedup, reduces DRAM access by 44∼50% and DRAM row activation by 41∼82%, without lossing accuracy.

II. BACKGROUND

A. Graph Neural Network

Graph neural network is a kind of artificial neural networks that can process data representing as graphs. Using notations in Table I, one layer of GNN can be formulated as follows:

$$h^{k+1}_u = \phi\left(x^k_u, \bigoplus_{v \in N^-_u} \psi(x^k_u, x^k_v, e^k_{v,u})\right),$$

where ψ and ϕ are message and update functions, respectively, and $\oplus$ is the aggregation operator. The aggregation phase roughly corresponds to $\oplus$ and ψ, while combination phase is the rest ϕ. For example, the Graph Convolutional Network (GCN) [21] employs neighbor feature with coefficient as ψ, sum as $\oplus$, and matrix multiplication as ϕ. The Graph Attention Network (GAT) [25] uses a self-attention coefficient in ψ, and a multi-head attention mechanism in $\oplus$ and ϕ.

Real-world graphs are extremely sparse and irregular. We define neighbor access irregularity for a traversal path as the mean of vertex index difference through it. Table II demonstrates basic metrics, sparsity η and irregularity ξ of a sequential traversal path with two mean types (arithmetic and geometric) on several datasets. It is clear that their sparsity are ultra high ($\eta > 0.9999$) and their irregularity ξ are just an order of magnitude lower than their number of vertices $|V|$.

TABLE II
GRAPH IRREGULARITY.

Graph	$\lvert V\rvert$	$\lvert E\rvert$	$1-\eta$	ξ_A	ξ_G
LiveJournal (LJ)	4.8e6	6.9e7	2.9e-6	7.9e5	4.7e5
Orkut (OR)	3.1e6	1.2e8	1.2e-5	8.1e5	5.8e5
Papers100M (PA)	1.1e8	1.6e9	1.3e-7	3.2e7	2.2e7

B. DRAM Hierarchy and Working Mechanism

In modern processor architecture, a memory controller communicates with memory modules over channels. A channel logically contains multiple memory modules (like DIMM), which in further consists of multiple DRAM chips. Following the hierarchy, rank, bank group, bank, row and finally column are introduced. The index of each hierarchy is directly specified by bits in given address, which the memory controller will provide on any read or write access. A row buffer acts as a placeholder when any row is desired, where the entire row is loaded and accessed in a smallest granularity called burst. Row buffer is exclusive to currently accessed row, which means accessing another row requires write-back and re-activation, which are slow and expensive.

Nowadays, modern architectures targeting neural networks mostly seek for efficient parallel execution. Accordingly, the DRAM bandwidth needs are urgent, hence they tend to employ many channels and maximize effective bandwidth of all channels by setting small interleaving and applying proper alignment. Such setup would scatter continuous address range to different rows, while keeping certain locality, which provides plenty room for our locality-aware hardware dropout.

III. MOTIVATION

A. Opportunity from GNN Robust Nature

GNNs are robust enough to tolerant certain loss of input information. Algorithmic dropout [7], [11], [14], [19] is a popular robustness application, which enhances model accuracy by masking some input data. Those algorithmic dropout efforts reach higher accuracy and better applicability, suggesting that not all memory accesses are indispensable, as accuracy remains unaffected. This insight highlights an opportunity for dropping irregular DRAM accesses in a locality-aware manner, to gain speedup without compromising model performance.

However, current GNN accelerators fail to take profit from robustness. The aggregation phase of GNNs is characterized by irregular memory access patterns [32], [34], posing significant challenges to execution platforms. Previous GNN accelerators [8], [15], [20], [29], [33] all assumed that every memory access during aggregation is essential, and have developed architectural optimizations based on this premise.

B. Inefficiency of Algorithmic Dropout from Burst View

Existing GNN robustness application mainly focus on accuracy enhancement, hence we first characterize if such approach will efficiently boost performance. All desired DRAM accesses finally resort to certain number of burst accesses, which is the minimal transaction in DRAM. Accordingly, we check if burst accesses are efficiently eliminated in aggregation with algorithmic dropout. We consider DRAM accesses in the aggregation phase, in a naive traversal path. We focus on the initial aggregation and ignore other phases and layers, on an architecture with one level LRU cache (hosts 4K features) and HBM. The normalized execution cycles, desired/actual DRAM access amount and total row activation amount for three datasets are shown in Figure 1(a)(b)(c), where the horizontal axis is drop rate from 0 to 1. We define data amount really used by aggregation (pass algorithmic dropout) as desired amount, and the triggered burst transaction amount as actual amount.

The results shown that under algorithmic dropouts from previous studies, the desired amount decreases almost linearly

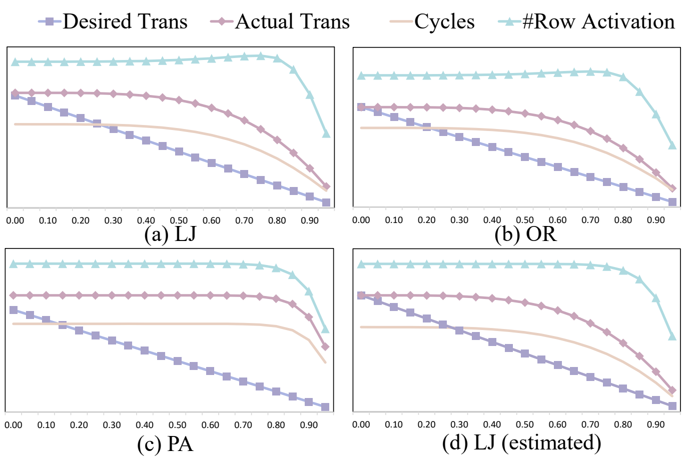

Fig. 1. Effect of Algorithmic Dropout on DRAM Access Metrics with Different Drop Rates (LRU cache with 4K item, naive traversal, HBM).

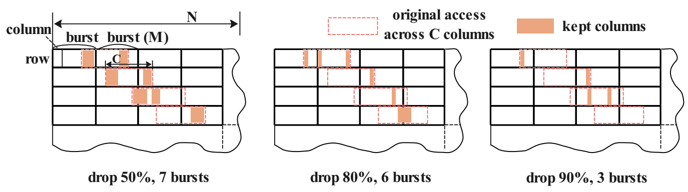

Fig. 2. Typical DRAM Structure and Dropout Example.

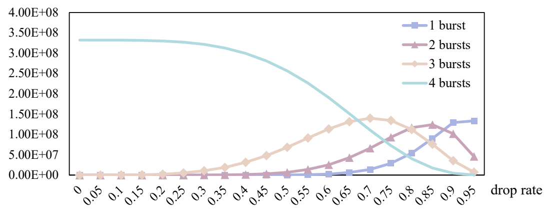

Fig. 3. Distribution of Burst Access Amount per Row Open Session.

as the drop rate increases, while the actual amount decreases much slowly. The total row activation amount is almost constant and finally reduces when drop rate $\alpha > 0.8$. In all, desired amount reflects algorithmic dropout linearly, but actual amount and row activation amount are not efficiently eliminated.

In all, we conclude that, current application of GNN robust nature, namely algorithmic dropout, cannot efficiently eliminate DRAM access and boost performance, due to their primary accuracy-improving goal and lack of hardware knowledge. A hardware-based solution that utilizes GNN robust nature to reach performance enhancement is necessary and promising.

C. Necessity and Benefit for Locality-aware Dropout

To estimate the theoretical speedup of locality-ware dropout over algorithmic dropout, we model how algorithmic dropout reflects on critical DRAM metrics like burst access or row activation amount. As shown in Figure 2, we assume a DRAM standard have N columns per row, M columns per burst, and K elements per burst, where M divides N. With interleaving, the input read accesses are Q random ones each covering C continuous columns in some row (typically $M < C < N$). We also assume the algorithmic dropout complies with Bernoulli(α) and ignore cache. Ideally, the desired DRAM access amount is $QC(1-\alpha)$. However, the possibility of whole burst drop is α^K, and the actual access amount will be $QC(1-\alpha^K)$. The inequality of drop rate to remaining bursts in Figure 2 shows the burst-minimal DRAM characteristic. Furthermore, the equivalent probability of a row skip is at most $\alpha^{CK/M}$, which can be ultra low. Based on the modeling to algorithmic dropout, we draw estimated metrics in Figure 1(d), which fits real values and demonstrates the correctness of the model.

On the other hand, if effective locality-aware dropout is proposed so that the actual access amount is direct proportion of kept rate (namely $1-\alpha$), its theoretical performance can be estimated. As for actual access amount, we expect algorithmic dropout to be $(1-\alpha^K)/(1-\alpha) = 1+\alpha+\cdots+\alpha^{K-1}$ times of locality-aware dropout. And for row activation amount, we may extend to $(1-\alpha^{CK/M})/(1-\alpha)$ times or even more, depending on different DRAM standards and interleaving.

With proper knowledge of DRAM organization and mapping, locality-aware dropout is totally practical, where burst access or row activation can be criteria of decision. Thus, we see an unexplored way to introduce burst or even row as hardware dropout granularity, to save the execution platforms from heavy and exhausting DRAM accesses. Row granularity dropout is aggressive compared to burst. Its gain is obvious, due to the reduce of expensive row activation, so does its impact, since a row can easily goes over 8KB on most DRAM standards, which can leads to too coarse granularity and even accuracy loss. However, thanks to the maximizing parallelism setup of modern neural network accelerators, the effective bursts per row open session is much lower than the row size, avoiding too coarse granularity in a single row. As shown in Figure 3, on Live Journal and GCN model with HBM, with alignment, the amount of actual access (burst) per row open session (max 4) is much less than number of bursts hosts in a row (64), which is normal due to graph irregularity and parallelism needs.

IV. Design of LiGNN

Guided by the above observations, we propose LiGNN, a data locality extraction and enhancement solution, which offers burst and row as brand-new dropout granularity in GNNs.

Figure 4(a) provides an illustration of the proposed architecture, where inside the red rectangle is our LiGNN. LiGNN does not depend on specific GNN training architectures, here we choose the SOTA GCNTrain-v3 [15] for example. The GCNTrain-v3 abstracts SpMM operation and implement it with separate datapath for sparse and dense matrices, where the sparse tiles belong to graph structure and dense tiles are from feature and weight matrices. LiGNN acts as an agent for dense requests and tiles back, without touching sparse ones. LiGNN will accept irregular read requests for dense matrix, conduct locality based filtering, send kept requests to and collect results from off-chip memory as usual, and finally output real accessed data and fake zero data to buffer of GCNTrain-v3 as dense tiles.

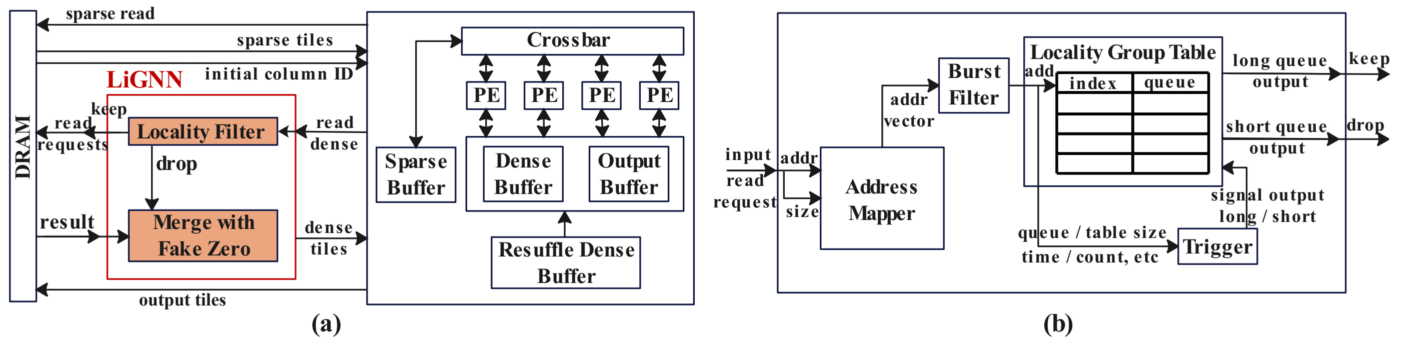

Fig. 4. Architecture of LiGNN based on GCNTrain.

The locality filter is the core component of LiGNN, which is shown in Figure 4(b). In LiGNN, algorithmic dropout is substituted by hardware, which is transparent for software.

Algorithm 1 locality_aware_group (burst dropout)

```
Input: read request stream $S$, burst filter $B$, trigger $F$
Output: request table $T$ with address vector index
while $S$ is not empty do
    $r \leftarrow S$.get_head_and_pop
    $a \leftarrow r$.address_range
    for each burst $b$ in $a$ do
        if $B$ drop $b$ then
            continue
        end if
        $v \leftarrow b$.address_vector
        $T[v] \leftarrow T[v] \cup \{b\}$
        notify $F$ with $T$.size, $T[v]$.size, etc.
        if $F$.fire then
            call locality_ordering_output
        end if
    end for
end while
```

A. Locality-aware Ordering with Burst Dropout

To handle massive DRAM read requests, we propose a locality-aware grouping and ordering policy to categorize and compare them efficiently. We use an address-vector-indexed table to temporarily store the grouped accesses for further decision-making, allowing for selection at the granularity of bursts or rows. When processing an incoming read request for a node feature, we first retrieve its address range from model information and generate the corresponding actual accesses (bursts) to that range, taking into account DRAM organization and mapping, with address translation applied if necessary. For each burst, we apply a filter B to determine whether it should be dropped, considering factors such as its effective ratio (since part of the burst may be masked by dropout) or load balancing.

After passing through the burst filter, burst accesses are grouped into the locality group table (LGT) based on their address vector, following a DRAM hierarchy. The LGT is implemented as a content-addressable memory (CAM), where the key is the row identifier and the value is a FIFO queue.

Algorithm 2 locality_ordering_output (row dropout)

```
Input: request table $T$ with address vector index
Input: desired size $n$, dropping rate $\alpha \in (0, 1)$, criteria $C$
Output: kept request queue $K$, dropped request queue $D$
Init: $\delta \leftarrow 0$
$k \leftarrow 0, d \leftarrow 0$
while $T$ is not empty and $k + d < n$ do
    if $\delta + (k + d)\alpha - d > 0$ then
        move shortest queue $x$ in $T$ to $D$
        $d \leftarrow d + x$.size
    else
        move the longest queue $x$ fits $C$ in $T$ to $K$
        $k \leftarrow k + x$.size
    end if
end while
$\delta \leftarrow \delta + (k + d)\alpha - d$
```

During the grouping process, a predefined custom trigger F is notified with relevant information such as the size of the LGT (or its items), elapsed time, or compute engine utilization. Based on this information, the trigger can decide when to output data. If no trigger is defined, all bursts that pass filter B are retained, while those that do not pass are dropped, rendering the LGT unnecessary in such cases.

B. DRAM Row Integrity Policy for Row Dropout

To reach row granularity dropout with custom trigger, a configurable drop rate $\alpha \in (0, 1)$ is required and a persist balance δ is maintained, whose sign bit determines whether next step is to drop or to keep. We also maintain a kept and dropped number in every call, for count and stop condition. If current state is to-drop, we drop the shortest queue in LGT, which is row granularity. Otherwise, we find the longest queue that fits custom criteria C and keep it. A random one is picked if multiple shortest/longest queues exist. Criteria C is set for needs like channel balancing or row-policy preference. For example, we can even cancel the queue size requirement and treat all queues equally. The process is ended when total output amount is reached and afterwards, the persist δ is updated. The comparison process of queue size is implemented in comparison complete binary tree style, where the values and indices are compared by trees to find large and small one (random if equal). Using the final output indices, we perform output operation over the corresponding FIFOs (queue in LGT).

C. Integration Discussion

How LiGNN integrates with other components is essential for practical usage. As for interface, besides adding dedicated signals, to utilize the QoS field of AXI protocol and a special QoS value as "droppable" is quite practical. If a bit droppable flag for every DRAM access is not feasible, the pre-configured address ranges are another way to distinguish between normal and droppable accesses. Sometimes the dropout mask is needed for other steps like backward pass, hence every address range or read request may also carry address info of its mask for LiGNN to write, otherwise sending flags back aside read results is also viable. The dropout mask is usually single-bit boolean and stored continuously, like an edge feature. Hence write to the mask will show good locality, in contrast to reading the feature data. The dropout scaling step, namely multiply by constant $1/(1-\alpha)$, is not done by LiGNN but done by compute unit, which can be easily inserted somewhere in the whole process.

TABLE III
DETAILED PARAMETERS FOR `LG-{A,B,R,S}`.

Name	Trigger Fire	Burst Filter	Row Filter	LGT size
`LG-A`	N.A.	Element-wise	N.A.	N.A.
`LG-B`	N.A.	Yes	N.A.	N.A.
`LG-R`	Every Feature	Optional	Yes	16x16
`LG-S`	Custom Interval	Optional	Yes	64x32

We implement several variants of LiGNN and name them by suffices `A`, `B`, `R`, `S`, while the prefix is `LG` for abbreviation of LiGNN. We first define `LG-A` as algorithmic dropout baseline, just for comparison. We then define `LG-B` as burst filter only version, where actually the LGT and trigger F do not exist. We also define `LG-R` as row filter version, namely LGT with trigger F and no burst filter, where the trigger fires on every feature read request. Finally, we expand the row filter scheduling range at decision (or trigger firing interval) to custom interval like certain number of features or by utilization of other units, and name it as `LG-S`. In all, from `LG-B`,`LG-R` to `LG-S`, the complexity and gain both increase. The detailed parameters for `LG-{A,B,R,S}` is shown in Table III.

V. EVALUATION

A. Methodologies

1) Evaluation Tools: Based on SOTA GCN training accelerator GCNTrain-v3, we design and implement a cycle-accurate simulator to measure the execution time in number of cycles. The critical component, DRAM, is simulated by Ramulator [12], which supports cycle-accurate model and energy estimation for various DRAM standards. To measure critical paths in LiGNN, we implement the core component, namely LGT with input/output logic in RTL and synthesize it in Verilog. We use the Synopsys Design Compiler with the TSMC 12 *nm* standard VT library for the synthesis and estimate the power consumption using Synopsys PrimeTime PX. The critical path,

TABLE IV
TYPICAL SPECIFICATIONS OF COMMON DRAM STANDARDS.

Standard	Frequency (MHz)	Bandwidth (GB/s)	Columns Per Row	Column Size (bits)	Burst
DDR3	400–1066	Up to 17	1K	64	8
DDR4	1600–3200	Up to 25.6	1K	64	8
GDDR5	1750–2000	Up to 256	1K	32	8/16
GDDR6	2500–3500	Up to 768	1K	32	16
LPDDR4	1600–4266	Up to 34	1K	64	16
LPDDR5	2750–6400	Up to 51.2	1K	64	16
HBM	500	128	128	128	2
HBM2	1000–1200	Up to 307	64	128	2

which is in CAM lookup, has a delay of 0.81 *ns* including the setup and hold time, enabling smooth executing of LiGNN at 1 GHz clock frequency. The access latency, energy, and area of on-chip memory like CAM and FIFO are estimated using Synopsys DesignWare Memory Compiler.

2) Baseline and System Configurations: We choose `LG-A` as baseline and `LG-{B,R,S}` as gradually improved design. The burst filters employ distribution in previous algorithmic dropout works. We choose HBM, DDR4 and GDDR5 as three representative DRAM standards, from various ones shown in Table IV. We focus on execution cycles, desired/actual DRAM access amount and total row activation amount.

3) Workloads: We choose the famous GCN [21], GraphSAGE [9] and GIN [28] with two layers as three main models, since the base architecture GCNTrain is for GCN training. The dataset used is listed in Table II. We select a dropout probability α from 0 to 1 with a step of 0.1.

B. Overall Results

1) Speedup: Figure 5 compares the speedup of the proposed `LG-{B,R,S}` and baseline `LG-A` over non-dropout value. We see that as α goes up, baseline `LG-A` reaches little speedup while `LG-{B,R,S}` achieve super-linear speedup, demonstrating the effect of locality-aware dropout.

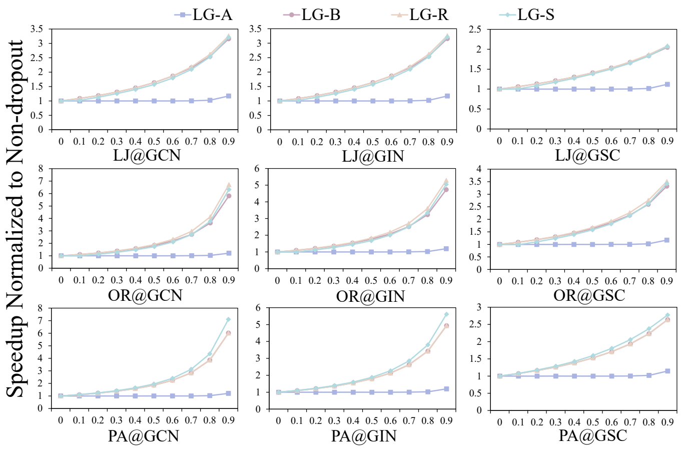

Fig. 5. Speedup on Different Droprates.

2) DRAM Access Amount: Figure 6 compares the DRAM access amount over non-dropout value on LJ dataset. As α goes up, baseline `LG-A` reduces little DRAM access, while

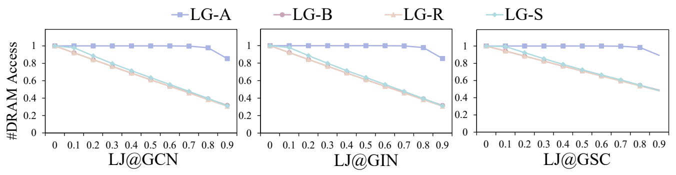

Fig. 6. Normalized Actual DRAM Access Amount on Different Droprates.

LG-{B,R,S} achieve linear reduction, showing the effect of DRAM burst granularity dropout.

3) Data Locality: DRAM row activation reflects the data locality via its total amount and consumes palpable energy. Figure 7 compares the DRAM row activation amount over different droprates. We see that LG-{A,B,R,S} successively expose smaller row activation amount, which means better locality and lower energy consumption. Specifically, LG-R is nearest to a linear correlation of α while LG-S is a little super-linear, revealing the effect of DRAM row integrity policy.

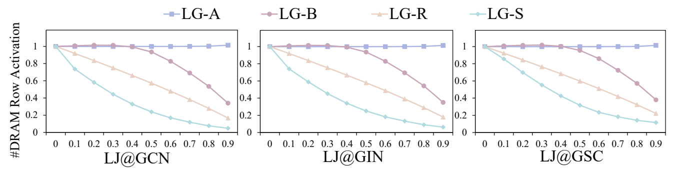

Fig. 7. Normalized DRAM Row Activation Amount on Different Droprates.

4) Area and Power: The LGT is essentially CAM+FIFO and it has the area of about 0.006 and 0.03 mm^2 with the power of at most 3 and 15 mW on LG-{R,S} respectively. Compared with the original area and power of GCNTrain (0.9mm^2 and 143 mW under TSMC 28nm) and other GNN accelerators, LiGNN consumes only a fraction and has a small overhead.

C. Exploration

We evaluate LiGNN on DDR4 and GDDR5 with GCN model. Figure 8 and Figure 9 compare speedup, DRAM access and row activation amount respectively, and are both similar to that on HBM, showing the good adaptability of LiGNN.

Numerous algorithmic dropout works have demonstrated that proper dropout will not degrade model accuracy, but help reach higher accuracy. We also analyze the effect of burst or row dropout in LiGNN on model accuracy using PyG and DGL, with a two-layer GCN model. As shown in Table V, burst or row dropout show no significant loss of model accuracy.

TABLE V
EFFECT OF BURST AND ROW DROPOUT ON MODEL ACCURACY.

Droprate	0	0.1	0.2	0.5
Burst Dropout	0.77	0.758	0.764	0.757
Row Dropout	0.77	0.76	0.768	0.762

VI. RELATED WORK

1) GNN Accelerators: Existing GNN accelerators [3], [5], [8], [15], [20], [24], [29]–[31], [33] reach great success in

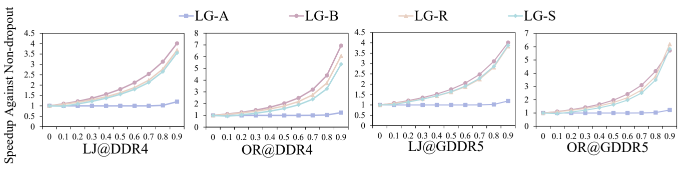

Fig. 8. Speedup over DDR4 and GDDR5.

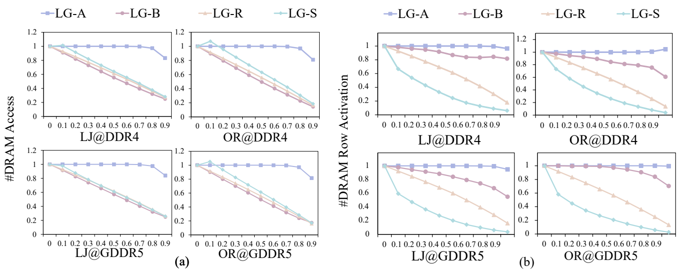

Fig. 9. DRAM Access and Row Activation Amount over DDR4 and GDDR5.

boosting performance of GNN execution and reduce energy consumption. However, they all operate under the assumption that every DRAM access during aggregation is essential, and have developed architectural optimizations based on this premise. Recent GAT acceleration work [10], [18] skip computation and access for low-attention value vertices, but such approximate computing approach still originates from algorithm improvement but not hardware demands.

2) GNN Algorthmic Dropout: GNNs are robust enough that they can tolerate proper loss of information, which has been utilized by various algorithmic efforts. Random dropping is firstly introduced in [11], and proved effective by works [1], [4], [23]. Work [2] proves that corrupted features are equivalent to L2-type regularization. Work [26] shows deeper and more clear relationship between dropout regularizer and L2 regularizer. As GNN comes up, random dropping has been generalized to graph data, by several representative works. DropOut [11], DropNode [14], DropEdge [19], DropMessage [7] propose dropout in different granularity from element to whole feature and successfully improve the model accuracy.

However, random dropping methods are purely algorithmic solutions aimed primarily at improving accuracy rather than enhancing performance. While they may incidentally reduce desired DRAM access, their lack of memory-awareness limits their effectiveness in minimizing actual DRAM access. Therefore, a hardware-based dropout solution is essential for accelerating GNN training.

VII. CONCLUSION

This work identifies an opportunity for improving data locality and gain speedup, with reasonable use of GNN robust nature. It proposes LiGNN, a hardware-based locality-aware dropout solution that can accelerate GNN training. Experimental results show that under classic 0.5 droprate, LiGNN achieves 1.6∼2.2x speedup, reduces 44∼50% DRAM access and 41∼82% DRAM row activation amount.

References

[1] Y. S. Abu-Mostafa. Learning from hints in neural networks. *J. Complex.*, 6(2):192–198, jun 1990.

[2] Chris M. Bishop. Training with noise is equivalent to tikhonov regularization. *Neural Computation*, 7(1):108–116, 1995.

[3] Dan Chen, Haiheng He, Hai Jin, et al. Metanmp: Leveraging cartesian-like product to accelerate hgnns with near-memory processing. In *Proceedings of the 50th Annual International Symposium on Computer Architecture*, ISCA '23, New York, NY, USA, 2023. Association for Computing Machinery.

[4] Ning Chen, Jun Zhu, Jianfei Chen, and Bo Zhang. Dropout training for support vector machines. In *Proceedings of the Twenty-Eighth AAAI Conference on Artificial Intelligence*, AAAI'14, page 1752–1759. AAAI Press, 2014.

[5] Xiaobing Chen, Yuke Wang, Xinfeng Xie, Xing Hu, Abanti Basak, Ling Liang, Mingyu Yan, Lei Deng, Yufei Ding, Zidong Du, and Yuan Xie. Rubik: A hierarchical architecture for efficient graph neural network training. *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, 41(4):936–949, 2022.

[6] Xinlei Chen, Li-Jia Li, Li Fei-Fei, and Abhinav Gupta. Iterative visual reasoning beyond convolutions. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pages 7239–7248, 2018.

[7] Taoran Fang, Zhiqing Xiao, Chunping Wang, Jiarong Xu, Xuan Yang, and Yang Yang. Dropmessage: Unifying random dropping for graph neural networks. *Proceedings of the AAAI Conference on Artificial Intelligence*, 37(4):4267–4275, Jun. 2023.

[8] Tong Geng, Ang Li, Runbin Shi, et al. Awb-gcn: A graph convolutional network accelerator with runtime workload rebalancing. In *2020 53rd Annual IEEE/ACM International Symposium on Microarchitecture (MICRO)*, pages 922–936, 2020.

[9] Will Hamilton, Zhitao Ying, and Jure Leskovec. Inductive representation learning on large graphs. *Advances in neural information processing systems*, 30, 2017.

[10] Dengke Han, Meng Wu, Runzhen Xue, Mingyu Yan, Xiaochun Ye, and Dongrui Fan. Ade-hgnn: Accelerating hgnns through attention disparity exploitation. In *Euro-Par 2024: Parallel Processing - 30th International Conference on Parallel and Distributed Computing, Madrid, Spain, August 25 - August 30, 2024, Proceedings*, Lecture Notes in Computer Science, pages 91–106, 2024.

[11] Geoffrey E. Hinton, Nitish Srivastava, Alex Krizhevsky, Ilya Sutskever, and Ruslan R. Salakhutdinov. Improving neural networks by preventing co-adaptation of feature detectors, 2012.

[12] Yoongu Kim, Weikun Yang, and Onur Mutlu. Ramulator: A fast and extensible dram simulator. *IEEE Computer Architecture Letters*, 15(1):45–49, 2016.

[13] Haiyang Lin, Mingyu Yan, Xiaochun Ye, Dongrui Fan, Shirui Pan, Wenguang Chen, and Yuan Xie. A comprehensive survey on distributed training of graph neural networks. *Proceedings of the IEEE*, 2023.

[14] Meng Liu, Hongyang Gao, and Shuiwang Ji. Towards deeper graph neural networks. In *Proceedings of the 26th ACM SIGKDD International Conference on Knowledge Discovery & Data Mining*, KDD '20, page 338–348, New York, NY, USA, 2020. Association for Computing Machinery.

[15] Heng Lu, Zhuoran Song, Xing Li, Naifeng Jing, and Xiaoyao Liang. Gcntrain: A unified and efficient accelerator for graph convolutional neural network training. In *2022 IEEE 40th International Conference on Computer Design (ICCD)*, pages 730–737, 2022.

[16] Y. Ma, H. Ren, B. Khailany, H. Sikka, L. Luo, K. Natarajan, and B. Yu. High performance graph convolutionai networks with applications in testability analysis. In *2019 56th ACM/IEEE Design Automation Conference (DAC)*, pages 1–6, June 2019.

[17] Lange Oliver and Perez Luis. Traffic prediction with advanced graph neural networks.

[18] Naebeom Park, Daehyun Ahn, and Jae-Joon Kim. Workload-balanced graph attention network accelerator with top-k aggregation candidates. ICCAD '22, New York, NY, USA, 2022. Association for Computing Machinery.

[19] Yu Rong, Wenbing Huang, Tingyang Xu, and Junzhou Huang. Dropedge: Towards deep graph convolutional networks on node classification. In *International Conference on Learning Representations*, 2020.

[20] R. Sarkar, S. Abi-Karam, Y. He, L. Sathidevi, and C. Hao. Flowgnn: A dataflow architecture for real-time workload-agnostic graph neural network inference. In *2023 IEEE International Symposium on High-Performance Computer Architecture (HPCA)*, pages 1099–1112, Los Alamitos, CA, USA, mar 2023. IEEE Computer Society.

[21] Franco Scarselli, Marco Gori, Ah Chung Tsoi, Markus Hagenbuchner, and Gabriele Monfardini. The graph neural network model. *IEEE Transactions on Neural Networks*, 20(1):61–80, 2009.

[22] Michael Schlichtkrull, Thomas N Kipf, Peter Bloem, Rianne Van Den Berg, Ivan Titov, and Max Welling. Modeling relational data with graph convolutional networks. In *European Semantic Web Conference*, pages 593–607. Springer, 2018.

[23] Nitish Srivastava, Geoffrey Hinton, Alex Krizhevsky, Ilya Sutskever, and Ruslan Salakhutdinov. Dropout: a simple way to prevent neural networks from overfitting. *J. Mach. Learn. Res.*, 15(1):1929–1958, jan 2014.

[24] Gongjian Sun, Mingyu Yan, Duo Wang, Han Li, Wenming Li, Xiaochun Ye, Dongrui Fan, and Yuan Xie. Multi-node acceleration for large-scale gcns. *IEEE Transactions on Computers*, 71(12):3140–3152, 2022.

[25] Petar Veličković, Guillem Cucurull, Arantxa Casanova, Adriana Romero, Pietro Liò, and Yoshua Bengio. Graph attention networks, 2018.

[26] Stefan Wager, Sida Wang, and Percy Liang. Dropout training as adaptive regularization. In *Proceedings of the 26th International Conference on Neural Information Processing Systems - Volume 1*, NIPS'13, page 351–359, Red Hook, NY, USA, 2013. Curran Associates Inc.

[27] Zonghan Wu, Shirui Pan, Fengwen Chen, Guodong Long, Chengqi Zhang, and S Yu Philip. A comprehensive survey on graph neural networks. *IEEE transactions on neural networks and learning systems*, 32(1):4–24, 2020.

[28] Keyulu Xu, Weihua Hu, Jure Leskovec, and Stefanie Jegelka. How powerful are graph neural networks? *arXiv preprint arXiv:1810.00826*, 2018.

[29] Runzhen Xue, Dengke Han, Mingyu Yan, Mo Zou, Xiaocheng Yang, Duo Wang, Wenming Li, Zhimin Tang, John Kim, Xiaochun Ye, and Dongrui Fan. Hihgnn: Accelerating hgnns through parallelism and data reusability exploitation. *IEEE Transactions on Parallel and Distributed Systems*, 35(7):1122–1138, 2024.

[30] Runzhen Xue, Mingyu Yan, Dengke Han, Zhimin Tang, Xiaochun Ye, and Dongrui Fan. Sihgnn: Leveraging properties of semantic graphs for efficient hgnn acceleration, 2024.

[31] Runzhen Xue, Mingyu Yan, Dengke Han, Yihan Teng, Zhimin Tang, Xiaochun Ye, and Dongrui Fan. Gdr-hgnn: A heterogeneous graph neural networks accelerator frontend with graph decoupling and recoupling. In *2024 61th ACM/IEEE Design Automation Conference (DAC)*, pages 1–6, 2024.

[32] Mingyu Yan, Zhaodong Chen, Lei Deng, et al. Characterizing and understanding gcns on gpu. *IEEE Computer Architecture Letters*, 19(1):22–25, 2020.

[33] Mingyu Yan, Lei Deng, Xing Hu, et al. Hygcn: A gcn accelerator with hybrid architecture. In *2020 IEEE International Symposium on High Performance Computer Architecture (HPCA)*, pages 15–29. IEEE, 2020.

[34] Mingyu Yan, Mo Zou, Xiaocheng Yang, et al. Characterizing and understanding hgnns on gpus. *IEEE Computer Architecture Letters*, 21(2):69–72, 2022.

[35] Rex Ying, Ruining He, Kaifeng Chen, Pong Eksombatchai, William L Hamilton, and Jure Leskovec. Graph convolutional neural networks for web-scale recommender systems. In *Proceedings of the 24th ACM SIGKDD International Conference on Knowledge Discovery & Data Mining*, pages 974–983, 2018.