2025 Design, Automation & Test in Europe Conference (DATE 2025)

GLEAM: Graph-based Learning through Efficient Aggregation in Memory

Andrew McCrabb
Computer Science and Engineering
University of Michigan
Ann Arbor, MI, USA
mccrabb@umich.edu

Ivris Raymond
Computer Science and Engineering
University of Michigan
Ann Arbor, MI, USA
ivris@umich.edu

Valeria Bertacco
Computer Science and Engineering
University of Michigan
Ann Arbor, MI, USA
valeria@umich.edu

***Abstract*—Graph Neural Networks (GNNs) have emerged as a powerful tool for analyzing relationship-based data, such as those found in social networks, logistics, weather forecasting, and other domains. Inference and training with GNN models execute slowly, bottlenecked by limited data bandwidths between memory and GPU hosts, as a result of the many irregular memory accesses inherent to GNN-based computation. To overcome these limitations, we present GLEAM, a Processing-in-Memory (PIM) hardware accelerator designed specifically for GNN-based training and inference. GLEAM units are placed per-bank and leverage the much larger, internal bandwidth of HBMs to handle GNNs' irregular memory accesses, significantly boosting performance and reducing the energy consumption entailed by the dominant activity of GNN-based computation: neighbor aggregation. Our evaluation of GLEAM demonstrates up to a 10x speedup for GNN inference over GPU baselines, alongside a significant reduction in energy usage.**

I. Introduction

Machine Learning (ML) models have revolutionized multiple industries, including healthcare, finance, manufacturing, logistics, and more. Conventional ML models (*e.g.*, linear regression, traditional neural networks, *etc.*) have provided the greatest value gains within domains characterized by vast, well-structured datasets, where each data instance (*e.g.*, an image or a row in a table) represents a single, independent data point.

However, many real-world industries depend not only on the *features* of data points, but also on the *relationships between* them. As examples, social networks comprise relationships between individuals, road networks are the road connections between intersections, e-commerce businesses model "bought-by", "sold-with", and other sets of relationships, and even weather can be predicted using the relationships of atmospheric and geographic conditions [1]. Such relationship-based data is most efficiently represented as a *graph*: a collection of vertices connected by edges that, together, map all elements and relationships between those elements.

Conventional ML models operate on independent data points and are ill-suited for extracting relational information that graph datasets contain. In contrast, *Graph Neural Networks* (GNNs), specifically designed to capture and utilize the dependencies between interconnected vertices, bridge the gap between the strengths of ML and graph-based data [2]–[4]. GNNs structure their neuron connections to directly mimic the edges of the graph, propagating information through interconnected vertices in each neural network layer. Figure 1 shows an example of an application and a portion of a GNN model based on that graph, where the network structure is based on the structure of the corresponding road network. With this graph-informed design, GNNs excel at training on and classifying in domains with rich relational information inherent to the application.

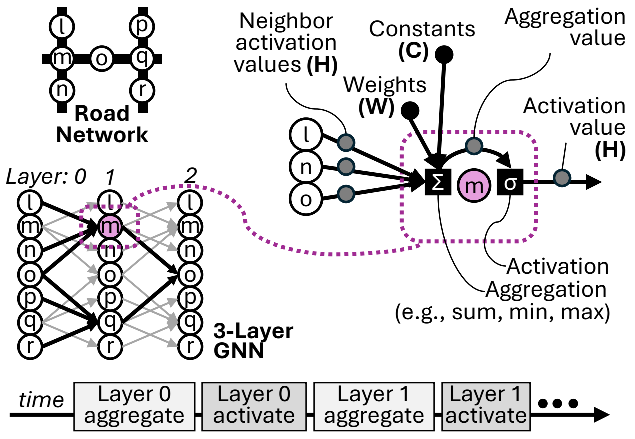

Fig. 1. An example road network, the structure of a corresponding 3-layer graph neural network with highlighted message-passing connections that affect vertex "o", and a detailed view of the per-vertex operations.

This capability yields immense real-world value. Social media companies use GNNs extensively to predict users' behaviors, recommend multimedia content, and optimize user-targeted advertising [5]. Video and other streaming services like Youtube and Netflix rely on video recommendation systems to personalize each video recommendation, which in turn often rely on GNNs [6]. Google uses GNNs to improve traffic predictions and optimal vehicle routing by 20-50% over other techniques [7]–[9]. GNNs are even used to monitor social media influence and misinformation spread detection [10] [11], weather forecasting [1], and more.

Although GNNs clearly offer substantial value, GNN-based operations are computationally expensive and energy inefficient. Further, conventional hardware accelerators (*i.e.*, GPUs and TPUs) offer minimal performance gains over CPUs and entail high power consumption, largely because GNN-based workloads involve many irregular memory accesses - accesses to non-sequential data that cannot be easily prefetched before the data is needed (*e.g.*, data from a vertex's neighbors in a graph) - which GPUs and TPUs are ill-equipped to handle [12]–

[14]. Even when paired with High Bandwidth Memory (HBM) devices, these irregular memory accesses lead to inefficient memory utilization, which bottlenecks performance and hinders scalability. As described in detail in Section II-A, these irregular memory accesses occur during "ForwardPass" operations, an essential step for both GNN training and inference. One family of specialized hardware solutions that has gained traction in applications that are bottlenecked by memory bandwidth is *Processing-in-Memory* (PIM) architectures: compute units placed within the memory device itself to take advantage of the greater bandwidth *internal* to the memory device.

To address these memory bandwidth bottlenecks in GNN operations, we present GLEAM, the first PIM hardware accelerator designed to leverage the high internal bandwidth of HBM devices. By processing data directly within memory, GLEAM reduces the inefficiencies of irregular memory accesses and accelerates aggregation, a core operation of GNN workloads. In summary, we make the following contributions:

- We characterize GNN model performance on CPU and GPU, noting that the sheer volume of irregular memory accesses entailed by loading vertex-specific values is the key performance bottleneck to "ForwardPasses" for GNNs, an essential part of both training and inference.
- We present GLEAM, a PIM-based, hardware accelerator for HBM memory devices that increases the throughput of *aggregation computations*, the most memory-intensive and time-consuming functions of GNN applications. Computing aggregation values in memory obviates much of the need to use the HBM-Host memory interface, increasing the throughput of aggregations by up to 17.7x.
- We evaluate GLEAM and demonstrate that it improves the performance of GNN inference by up to 10x compared to a GPU baseline, while consuming 77% less energy, on average, per inference versus a GPU.

This paper is organized as follows. In Section II, we describe background material. We present GLEAM, our PIM hardware accelerator for GNN workloads, in Section III. Section IV presents our evaluation of GLEAM against GPU baselines, while Section V summarizes related work.

II. Background

A. Graph Neural Networks

Graph Neural Networks (GNNs) are a specialized class of neural networks grounded on the graph structure of their input. Like other NN models, GNNs consist of multiple message-passing layers. However, unique to GNNs, the layers' topology, and thus the overall model's topology, is derived from the input graph itself. Specifically, the nodes comprising each layer correspond to vertices in the original graph: each vertex i in the graph is associated with a node n_i^l at layer l, and these nodes exchange information with their neighbors to capture the structural properties of the graph.

With respect to the right part of Figure 1, each node n_i^l in each layer l outputs an activation value h_i^l, collectively forming a vector H^l of activation values for that layer. The calculation of a single h_i^l, visualized on right side of Figure 1, involves an **aggregation function** with three input vectors: edge-specific vector of constants[1] $\mathbf{C}$, layer-specific weights $\mathbf{W}^l$, and the activation values of the previous layer $\mathbf{H}^{l-1}$. Typical aggregation functions include min, max, and $mean$. The constants $c_{i,j}$ in C capture the structural relationship between vertex i and its neighbor j. Note that each h_i^l may, itself, be a vector, with elements representing the features of the application; for example, in a weather prediction application, h_i^l vectors might include features like humidity, temperature, and wind speed. These features contained in h_i^l coming into or out of a layer are called the input features and output features, respectively, and they have different cardinalities. The size of W^l is, in turn, the product of the input and output features. For example, if a layer has 10 input and 8 output features, W^l contains 80 values.

The specific calculation of $c_{i,j}$ varies by GNN implementation, as shown in Table I. The result from aggregation is the input to an **activation function**. This activation function is typically *ReLU*, but other options include *LeakyReLU*, *sigmoid*, and *tanh* functions [2], [3].

$$h_i^l = \sigma\left(\sum_{j\in J}\frac{1}{c_{i,j}}h_j^{l-1}W^l\right) \tag{1}$$

Equation 1 describes the aggregation and activation for each node where J is the set of vertex i's neighbors and $\sum$ and σ denote the aggregation and activation functions, respectively.

During GNN training, both ForwardPass and BackwardPass operations are executed. The **ForwardPass** propagates input features through the layers as described above to generate classifications, while the **BackwardPass** computes gradients to update the model's weights W. In contrast, inference requires only the ForwardPass to produce classifications. This paper focuses on optimizing the ForwardPass, as it is both the primary step for inference and a dominant part of training.

Algorithm 1: Graph Neural Network Inference

```
# ForwardPass
For each layer l in L
    For each node i in l with J neighbors
        $h_i^l$ = Activate(Aggregate($C, H_j^{l-1}, W_{j,i}^l$))
```

This work considers the three most common models for GNNs representing the current state of the art: GraphSAGE (GS) [2], Graph Convolution Networks (GCN) [3], and Graph Attention Networks (GAT) [4]. The primary difference between these models is the function used in calculating C. Table I outlines the specific functions adopted by each of the three types of GNN models.

A key distinction between GNNs and other NNs is the network's density: the proportion of connections between layers against the total number of possible connections. Density is the inverse of sparsity. In fully-connected NN layers, each node connects to every other node, resulting in a density

[1]Though described as constants in prior works, $\mathbf{C}$ values in Graph Attention Network models [4] are trainable vectors like W and α.

Model	$c_{i,j} =$
GAT	$\sigma\left(\text{LeakyReLU}\left(\vec{a}^T((W_i^h\|\|(W_j^h))\right)\right)$
GCN	$\sqrt{\|h_i\|}\sqrt{\|h_j\|}$
GraphSAGE	$\|h_i\|$

TABLE I
CALCULATIONS FOR EDGE-SPECIFIC CONSTANTS C ACROSS DIFFERENT GNN IMPLEMENTATIONS

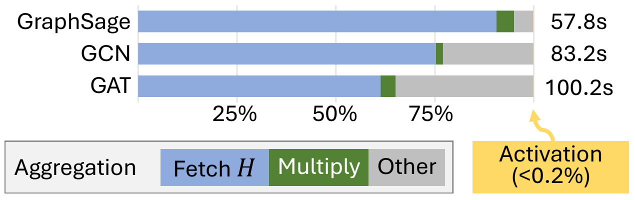

Fig. 2. ForwardPass CPU execution breakdown from VTune for GraphSAGE, Graph Convolutional Network, and Graph Attention Network models after 10 training iterations on the LiveJournal graph

of one. State-of-the-art pruning techniques can remove 30-90% of these connections, leading to densities between 0.7 and 0.1, respectively [15] [16]. In contrast, GNNs, which base their connections on the inherently sparse graphs of the application, typically exhibit densities below 0.00001—five or more orders of magnitude smaller than typical NN models! If GNNs represented these connections with traditional, dense, tensor structures, >99.99% of their values would be '0', so GNNs instead use sparse data representations, like edgelists or adjacency lists to conserve storage space. However, while these sparse representations are necessary, they also create many additional *irregular* memory accesses to h_j, significantly impairing performance. Note that W is typically small (input features $\times$ output features) and C is accessed linearly, so H is the only part of Equation 1 responsible for the many irregular accesses, the effect of which is discussed below.

B. Memory Technologies

Memory-bound applications like GNNs demand high data bandwidth, beyond what traditional DDR memory devices provide. For such applications, data centers often use High Bandwidth Memory (HBM) devices, which offer greater bandwidth and lower power consumption compared to DDR memory devices [17]. However, even with HBM, GNN performance remains bottlenecked by the available memory bandwidth [18].

Processing in Memory (or "Compute in Memory") architectures further mitigate this bottleneck by integrating compute units directly within memory devices. This approach significantly reduces data movement between memory and CPU or GPU. While PIM architectures are prohibitively expensive for almost any company to design, manufacture, and use merely for their own internal needs, these expenses can be offset by deploying them through cloud services, similar to offerings like Google's TPU [19] and UPMEM's PIM [20].

III. GLEAM

A. Motivational Study

Despite their massive utility, GNN-based workloads are slow: a single inference on an industrial-sized graph can take an hour or more on either a CPU or a GPU [12] [13]

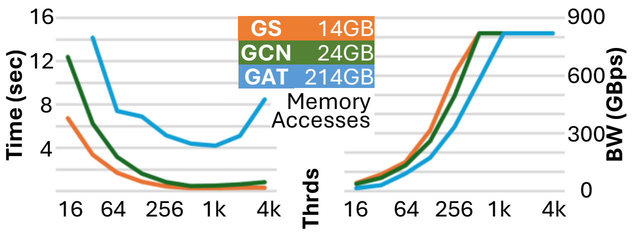

Fig. 3. Performance (left), data transferred between host and HBM memory (center), and memory bandwidth utilized (right) during inference, using GAT, GCN, and GraphSAGE models trained on the LiveJournal dataset across different parallel thread counts simulated on a 16K-core GPU with HBM3 memory.

[21]. To understand the key performance bottlenecks in GNN operations, we conducted a motivational study using Intel's VTune to analyze the ForwardPass computations, the only major function of an inference and a fundamental, dominant function of a training iteration, across the three most popular GNN models. We used an Intel i7-8700k with DDR4 memory using the LiveJournal dataset [22]. The results are shown in Figure 2. The ForwardPass comprises many aggregation and activation operations, but execution time is dominated by the aggregations. The aggregation function, detailed in Section II-A, multiplies and accumulates three sets of values: vertex values from the previous layer (H), layer-specific weights (W), and model-specific constants (C). While the multiply and accumulate operations consume little time, reading the values in H incurs many, irregular memory accesses: the required values for one aggregation operation are often stored in non-contiguous memory locations, leading to inefficient cache usage, where only a small fraction of a fetched cache line is used before the whole line is discarded. As a result, H accesses require much higher memory bandwidth than the fetching of W weights. To solidify our understanding, we consider several additional GNNs with more layers and input/output features: we observed an expected increase in overall runtime, but the proportion of time spent on each component of the execution remained substantially the same.

Given the challenge posed by limited memory bandwidth, we conducted a second motivational study to gain insights on how bandwidth saturation affects performance as the thread parallelism of the ForwardPass implementation increases. We used ZSim, a cycle-accurate microarchitectural simulator described further in Section IV, to model GNN inference on a GPU with 16K cores connected to HBM3 memory. Figure 3 presents the results of this study, showing that, beyond 512-1024 threads, further increases in parallelism do not yield additional performance improvements as memory bandwidth fully saturates. Note that >92% of the 16K GPU cores remain idle at peak GPU performance. These findings demonstrate that any effective acceleration solution for GNN-based workloads must increase available memory bandwidth and thus minimize delays from irregular memory accesses - insights that informed our GLEAM accelerator design.

Note that, due to the high memory bandwidth demands and poor GPU core utilization, both CPUs and GPUs have been shown to offer similarly poor ForwardPass performance [23]. As HBM devices are most commonly paired with GPU platforms, we use GPUs as the baseline against which to compare our proposed solution.

B. Accelerator Architecture

Our processing in memory accelerator, GLEAM, leverages the much higher bandwidth *inside* HBM3 memory to quickly aggregate data directly in memory – rather than relying solely on the much lower HBM-host bandwidth. That is, instead of sending all data to the host, the memory device only sends the aggregated results, substantially reducing the overall traffic between host and memory.

Figure 4 illustrates the architecture of the GLEAM hardware accelerator. We deploy one accelerator per bank, connected directly to the corresponding column decoder. GLEAM contains three functional units: a multiplier, an accumulator, and a comparator, with which GLEAM supports all the relevant Σ functions for the purpose of aggregation: SUM, MIN, MAX, and MEAN. These units all perform computation in the bFloat16 format, while addresses are 64-bit wide. GLEAM also contains three buffers to store the inputs to the aggregation step received from the host. The host interacts with GLEAM through four commands, three are needed to prepare GLEAM for aggregation setting up the three buffers, while one is used to issue the aggregation itself, synchronized across all GLEAM units in the system. Once the `address`(h_j), W^l, and $c_{i,j}$ buffers are filled, the host issues the `aggregate` command to specify which Σ function to use and to initiate aggregation. In the first step of aggregation, GLEAM fetches a neighbor of the current node in aggregation by issuing the corresponding `address`(h_j) to the address decoder within the bank (①). Note that each GLEAM unit has access only to the data available in its bank, so a small number of h values are replicated across multiple banks. As GLEAM receives the neighbor's previous activation value (h_j^{l-1}) from the column decoder (②), it sends the weight w_i^l and the appropriate $c_{i,j}$ from their respective buffers to the multiplier. After multiplication (③), GLEAM sends the product of these terms ($w_i^l \times h_j^{l-1} \times c_{i,j}$) to the unit implementing the Σ function for this algorithm (④). These steps compose a partial aggregation and are repeated for each neighbor of the aggregating node. Finally, if the Σ function is MEAN, the host divides the final value obtained by the number of neighbors considered during aggregation (not shown in Figure 4). To store the final aggregation result, GLEAM overwrites 16 bits of the row buffer (⑤), which the host will access during the subsequent activation step.

While both GLEAM and the GPU baseline are bottlenecked by the sparsity of a given dataset, GLEAM scales with the internal bandwidth (30TB/s) of HBM3, rather than the external bandwidth (819GB/s). The roofline model in Figure 5 presents the gap of opportunity this much higher bandwidth creates between the memory-bound performance of GLEAM and that of a baseline GPU. Datasets with fewer FLOPs per Byte are more sparse, leading to GLEAM's performance of up to 17.7× that of a GPU.

GLEAM does not introduce additional latency when performing aggregation: computation time (t_{GLEAM}) is masked

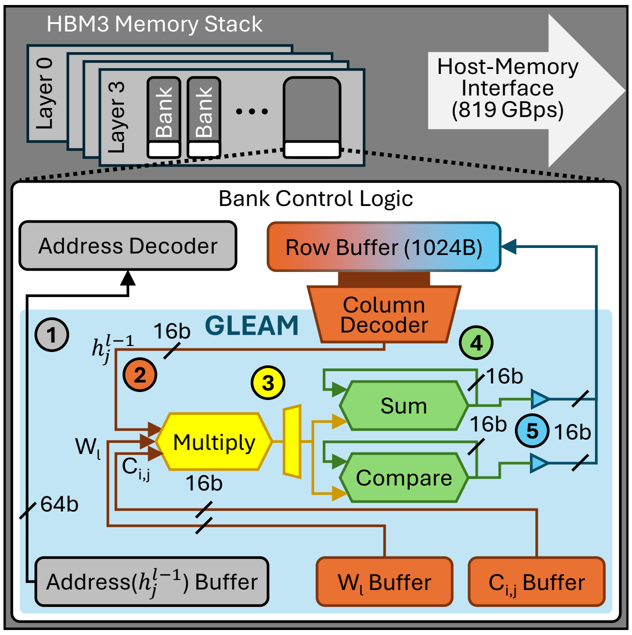

Fig. 4. GLEAM connects to the column decoder of each bank in the HBM memory stack. ① GLEAM accesses a neighbor's data from bank memory. ② The appropriate weight, constant, and neighbor data are sent to the multiplier. ③ GLEAM computes the product and forwards it to the accumulator or comparator, depending of the Σ function. ④ Steps 1-4 are repeated for all neighbors of the aggregation node. ⑤ GLEAM sends the final aggregation value to the row buffer, where the host can retrieve it.

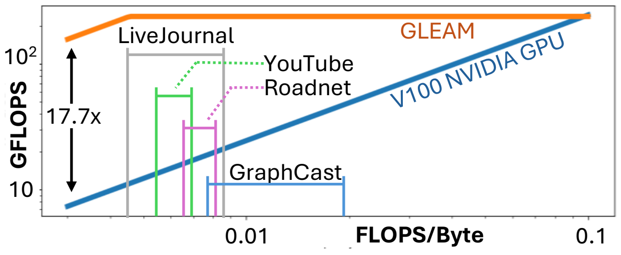

Fig. 5. Roofline model of GLEAM vs. a V100 NVIDIA GPU with HBM3 memory based on a GraphSAGE GNN implementation. Four benchmarks of varying in operational intensity (*i.e.*, FLOPs/Byte) are considered. The operational intensities of each benchmark vary with the quality of the underlying graph's partitioning in memory.

by the HBM row precharge time (t_{RP}). As can be noted in Figure 6, after GLEAM receives the necessary data from the column decoder, it immediately begins aggregation, while the bank precharges the previous row. This design bounds GLEAM's compute performance at t_{RP}/t_{GLEAM} partial aggregations per row.

IV. Experimental Evaluation

A. Experimental Setup

We evaluate GLEAM's performance against a GPU baseline. Both the baseline system and our GLEAM-augmented solution leverage a 16K-core GPU core paired with HBM3 memory. Our setup leverages a simulation framework that includes an extended version of ZSim (a cycle-accurate microarchitectural simulator [24]) and the Ramulator DRAM simulator [25] to accurately model GPU, HBM, and the GLEAM architectures.

We evaluate GLEAM by simulating inference on synthetic GNN datasets based on four real-world graph datasets: Live-

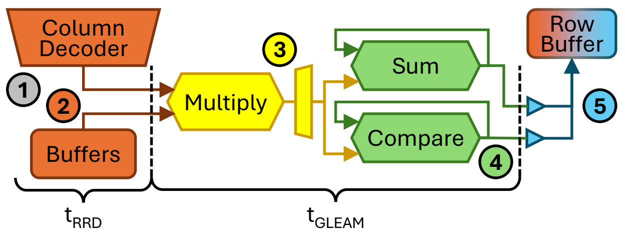

Fig. 6. ①, ② GLEAM loads the neighbors and weights into the multiplier, incurring the standard internal memory access latency t_{RRD}. ③, ④ GLEAM computes a partial aggregation (t_{GLEAM} latency), while the HBM bank performs row precharge.

Dataset	|V|	|E|	Degree	LR	Industry
LiveJournal (LJ)	4.8M	43.1M	17.79	1.56	Social media
RoadNet (RN)	2.0M	2.8M	2.82	2.16	Road navigation
YouTube (YT)	1.1M	3.0M	5.27	1.80	Video tecomm.
GraphCast (GC)	655k	2.6M	8.00	2.57	Weather forecast

TABLE II
GRAPH DATASETS AND THEIR PROPERTIES

Journal [22] connects social media accounts; Roadnet [26] connects road intersections; YouTube [27] connects users and videos; and GraphCast [1] connects sections of the Earth's atmosphere which form a multi-layer, global mesh to predict future weather conditions. Table II describes key characteristics of these datasets: vertex and edge count, average degree (*i.e.*, average number of neighbors), and average line reuse (LR): a measure of the graph's partitioning efficiency. Specifically, LR quantifies how many of a vertex's neighbors are stored within the same cache line. A higher LR value indicates that more of the H data that is used for the same node's computation is stored contiguously, leading to fewer memory accesses, better cache efficiency, and faster performance. For example, RoadNet and GraphCast have higher LR values because their graph structures are based on physically-organized units, making it easier to store neighboring vertices together in memory. In contrast, LiveJournal and YouTube, which represent social or recommendation networks, represent less cleanly-organized domains and have less predictable edges. This disorganization results in lower cache line reuse since neighbors are less likely to be stored close together in memory. Lower LR leads to worse baseline performance and affects GLEAM's potential performance gains. In order to compare GLEAM against strong, realistic baseline configurations, we utilize partitions provided by METIS, a state-of-the-art hierarchical partitioning tool.

B. *Performance*

Figure 7 shows the performance of GNN workloads running on GLEAM, compared to a GPU baseline, across different datasets and models. GLEAM achieves 2-8x speedup over the GPU, with the most significant gains for datasets with lower LR (*i.e.*, YouTube and LiveJournal), because the baseline is more pressured for memory bandwidth. We also observe that GLEAM offers higher performance gains for GCN than GraphSage (GS): GCN uses a more complex calculation for C for each vertex, so the GPU baseline experiences a larger runtime than GraphSage. In contrast, GraphSage and GCN workloads with GLEAM finish with nearly identical runtimes, as the extra work for GCNs is completed by the GPU and is

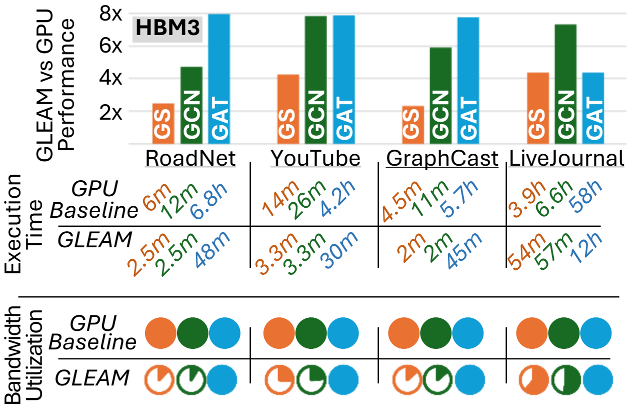

Fig. 7. Top: Relative performance comparison of GLEAM to a GPU baseline. Middle: Runtime for 100 inferences using 5-layer GNN models with 10 input and output features. Bottom: HBM3 bandwidth utilization as a percentage of full saturation (819 GBps).

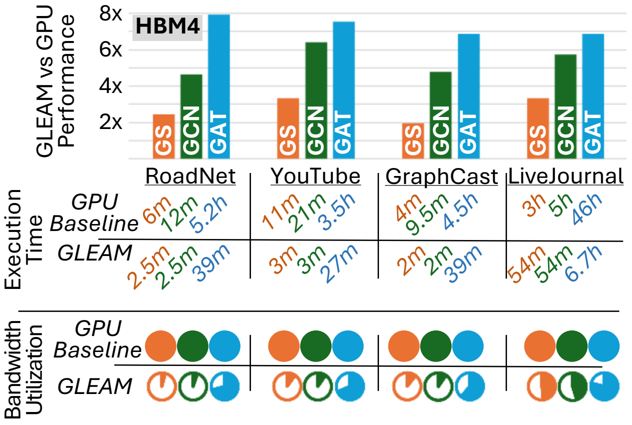

Fig. 8. Top: Relative performance comparison of GLEAM to a GPU baseline connected to a hypothetical HBM4. Middle: Runtime for 100 inferences using 5-layer GNN models with 10 input and output features. Bottom: Bandwidth utilization of a hypothetical future HBM4 device as a percentage of full saturation (1.5 TBps).

fully masked by the aggregations completed in the GLEAM units, resulting in higher relative performance.

GAT workloads require more complex computations and memory accesses to calculate C than GraphSAGE and GCN, as seen in Table I. Much of this computation can be masked by the latency GLEAM needs to complete its aggregations, enabling GLEAM to offer even higher performance gains vs the GPU than for GraphSage and GCN workloads in most cases. However, while the calculation for C can be fully masked by the aggregations for the GCN model workloads, it can only be partially masked for GAT models, largely because that computation requires significantly more data: per-vertex attention vectors. As a result, while the wide HBM3 memory bandwidth is fully saturated in all cases for the GPU baseline, that bandwidth remains fully saturated only for GAT workloads when using GLEAM, while other benchmarks show that GLEAM is capable of sufficiently reducing memory bandwidth pressure so to satisfy all the host's need.

Future HBM Devices. To better understand this impact, we evaluated GLEAM vs the GPU baseline when both systems

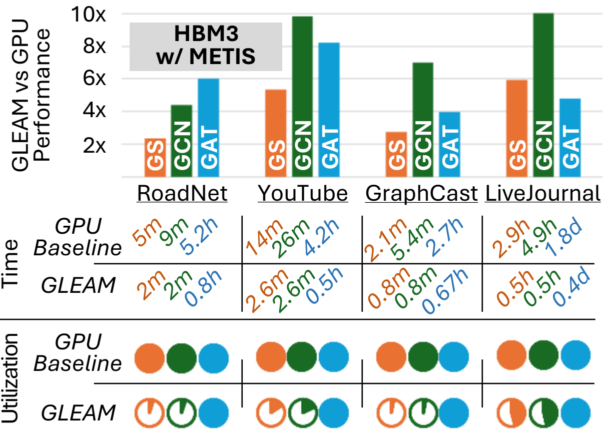

Fig. 9. Top: Relative performance of GLEAM to a GPU baseline when employing METIS graph partitioning in pre-processing. Middle: Runtimes for 100 inferences using 5-layer GNN models with 10 input and output features. Bottom: HBM3 bandwidth utilization as a percentage of full saturation.

Energy (kJ)	**GS**	**GCN**	**GAT**
GPU Baseline	**29.0**	**49.3**	**452.3**
- GPU	28.7	48.7	447.5
- HBM	0.31	0.53	4.76
GLEAM	**7.8**	**8.0**	**112.9**
- GPU	7.4	7.5	110.8
- HBM	0.12	0.21	1.84
- PIM	0.27	0.27	0.27
Energy Savings	**73.1%**	**83.7%**	**75.1%**

TABLE III
BREAKDOWN OF ENERGY USAGE FOR GLEAM AND THE GPU BASELINE ACROSS DIFFERENT MODELS USING THE LIVEJOURNAL DATASET

use a theoretical HBM4 with 1.5 TBps external bandwidth and doubled bank count. The results of this study are shown in Figure 8. Additional external memory bandwidth (1.5 TBps) improves slightly the GPU baseline performance, while additional memory banks increase GLEAM's performance significantly. Moreover, more of the GAT workloads' work to calculate their complex C is masked by now-offloaded aggregations. Indeed, we observe a 7-8x performance improvement of GLEAM vs. the GPU baseline for all GAT workloads.

Effect of Graph Partitioning. Besides memory bandwidth, the way vertex data is partitioned, measured by line reuse, also affects GNN workload speed. For some domains, a pre-computation step to partition vertex data is used to enhance memory bandwidth utilization, especially for latency-sensitive or high-throughput tasks [28] [29]. To assess the impact of graph partitioning on GLEAM, we carried out a study to partition per-vertex H data into cache lines for GPUs, and into HBM rows for GLEAM using METIS [30], the state-of-the-art graph partitioning tool. As shown in Figure 9, optimal partitioning alone provides a 1.45x speedup for our baseline GPU systems. With GLEAM, the performance improves by an *additional* 5.9x on average, and up to **10x** for GCN workloads, where GPU computations are more efficiently masked by in-memory aggregations.

C. Energy and Area

Energy. Table III reports the energy usage for one GNN-based inference for the GPU Baseline and GLEAM, using each of the three GNN models considered. Our analytical model uses metrics obtained from ZSim simulations with power estimates from the NVIDIA V100 datasheet [31]. For HBM, we use 1.2 pJ/bit for data transfers from subarray to row buffer and 3.7 pJ/bit for row buffer to host [32]. The energy consumption of each component in the GLEAM units: 16-bit multiplier, comparator, buffers, *etc.* are extrapolated from established benchmarks in prior studies [33]–[36]. We observe that GLEAM achieves an impressive 70-85% reduction in energy usage, compared to the GPU Baseline, averaging a 77.3% savings. This gain comes primarily by offloading work from the power-hungry GPU. The GLEAM units consume negligible energy, relative to the GPU host. They further decrease the HBM's energy consumption by reducing the volume of data transferred to the GPU.

Area. We estimated the area of GLEAM using the SKY130 PDK [37] and Synopsys Design Compiler®, applying scaling factors established from prior work [38]. Based on those analyses, GLEAM requires approximately 1966μm^2 at a 20nm process node, occupying only 0.28% of an HBM sub-array (~46 rows) per bank [39]. GLEAM does not, therefore, significantly impact an HBM's memory storage capacity.

V. Related Work

Several works have explored PIM approaches in addressing similar challenges to those posed by the irregular memory access patterns in GNN workloads, but not GNN workloads themselves. [40] and [41] proposed PIM-based accelerators using Hybrid Memory Cube (HMC) and UPMEM's DDR4 architectures, respectively, for sparse matrix-vector multiplication, highlighting the potential for PIM-based GNN acceleration on now-outdated baseline memory products. Further, [42] characterizes PIM performance for sparse matrix operations on UPMEM's DDR4 PIM offerings.

In addition, some works accelerate GNNs and similar workloads featuring many sparse memory operations with non-PIM approaches. [43], [44], and [45] focus on optimizing data reuse and organization in older HBM environments to accelerate sparse matrix multiplications. [46], [47], and [13] are non-PIM accelerators targeting GNN inference specifically, but are limited by the lower external memory bandwidths available for HBM and other DRAM devices. [48] and [49] present FPGA-based and DIMM-level sparse matrix multiplication accelerators. To the best of our knowledge, however, no prior work has proposed an HBM-based PIM accelerator for GNN workloads.

Conclusion

In this work, we presented GLEAM, a Processing-in-Memory (PIM) hardware accelerator designed to boost the performance of Graph Neural Network (GNN) workloads. GLEAM addresses the limitations of conventional accelerators, like GPUs and TPUs, in handling irregular memory accesses by leveraging the high internal bandwidth of High Bandwidth Memory (HBM) devices and a custom hardware design to accelerate GNN aggregations, the dominant bottleneck in GNN performance. Our evaluation demonstrates that GLEAM achieves up to a 10x speedup in GNN inference, compared to a GPU baseline, and a 77% reduction in energy consumption.

References

[1] R. Lam *et al.*, "Learning skillful medium-range global weather forecasting," *Science*, 2023.

[2] W. L. Hamilton, R. Ying, and J. Leskovec, "Inductive representation learning on large graphs," in *Proc. NIPS*, ser. NIPS'17, 2017.

[3] T. N. Kipf and M. Welling, "Semi-supervised classification with graph convolutional networks," 2017.

[4] P. Veličković, G. Cucurull, A. Casanova, A. Romero, P. Liò, and Y. Bengio, "Graph attention networks," 2018.

[5] N. Klingler, "Graph neural networks (gnns) – 2024 comprehensive guide," 2024. [Online]. Available: https://viso.ai/deep-learning/graph-neural-networks

[6] T. Rawka and M. Jadeja, "A survey on graph neural network based video recommendation system," in *Proc. CIKM*, 2022.

[7] W. Jiang, J. Luo, M. He, and W. Gu, "Graph neural network for traffic forecasting: The research progress," *ISPRS International Journal of Geo-Information*, 2023.

[8] W. Jiang and J. Luo, "Graph neural network for traffic forecasting: A survey," *Expert systems with applications*, 2022.

[9] A. Derrow-Pinion, J. She, D. Wong, O. Lange, T. Hester, L. Perez, M. Nunkesser, S. Lee, X. Guo, B. Wiltshire, P. W. Battaglia, V. Gupta, A. Li, Z. Xu, A. Sanchez-Gonzalez, Y. Li, and P. Velickovic, "Eta prediction with graph neural networks in google maps," in *Proc. CIKM*, 2021.

[10] H. T. Phan, N. T. Nguyen, and D. Hwang, "Fake news detection: A survey of graph neural network methods," *Applied Soft Computing*, 2023.

[11] L. Gao, H. Wang, Z. Zhang, H. Zhuang, and B. Zhou, "Hetinf: social influence prediction with heterogeneous graph neural network," *Frontiers in Physics*, 2022.

[12] A. Auten, M. Tomei, and R. Kumar, "Hardware acceleration of graph neural networks," in *2020 57th ACM/IEEE Design Automation Conference (DAC)*, 2020.

[13] K. Kiningham, P. A. Levis, and C. Ré, "Grip: A graph neural network accelerator architecture," *IEEE Transactions on Computers*, 2020.

[14] Y. Wang, B. Feng, Z. Wang, G. Huang, and Y. Ding, "Tc-gnn: Bridging sparse gnn computation and dense tensor cores on gpus," in *Proc. USENIX*, 2023.

[15] V. Sanh, T. Wolf, and A. Rush, "Movement pruning: Adaptive sparsity by fine-tuning," in *Advances in Neural Information Processing Systems*, 2020.

[16] H. Cheng, M. Zhang, and J. Q. Shi, "A survey on deep neural network pruning: Taxonomy, comparison, analysis, and recommendations," *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 2024.

[17] JEDEC, "Jedec standard: High bandwidth memory (hbm3) dram, jesd238a," 2023. [Online]. Available: https://www.jedec.org/standards-documents/docs/jesd238a

[18] Z. Wang, Y. Wang, C. Yuan, R. Gu, and Y. Huang, "Empirical analysis of performance bottlenecks in graph neural network training and inference with gpus," *Neurocomputing*, 2021.

[19] N. Jouppi *et al.*, "Tpu v4: An optically reconfigurable supercomputer for machine learning with hardware support for embeddings," in *Proc. ISCA*, 2023.

[20] B. Friesel, M. Lütke Dreimann, and O. Spinczyk, "A full-system perspective on upmem performance," in *Proceedings of the 1st Workshop on Disruptive Memory Systems*, ser. DIMES '23, 2023.

[21] L. Zhang, Z. Lai, Y. Tang, D. Li, F. Liu, and X. Luo, "Pcgraph: Accelerating gnn inference on large graphs via partition caching," in *Proc. BDCloud*, 2021.

[22] L. Backstrom, D. Huttenlocher, J. Kleinberg, and X. Lan, "Group formation in large social networks: membership, growth, and evolution," in *Proc. SIGKDD*, 2006.

[23] J. B. Park, V. S. Mailthody, Z. Qureshi, and W.-m. Hwu, "Accelerating sampling and aggregation operations in gnn frameworks with gpu initiated direct storage accesses," *Proc. VLDB Endow.*, 2024.

[24] D. Sanchez and C. Kozyrakis, "Zsim: Fast and accurate microarchitectural simulation of thousand-core systems," *ACM SIGARCH Computer architecture news*, 2013.

[25] Y. Kim, W. Yang, and O. Mutlu, "Ramulator: A fast and extensible dram simulator," *IEEE Comput. Archit. Let.*, 2015.

[26] J. Leskovec, K. J. Lang, A. Dasgupta, and M. W. Mahoney, "Community structure in large networks: Natural cluster sizes and the absence of large well-defined clusters," *Internet Mathematics*, 2009.

[27] J. Yang and J. Leskovec, "Defining and evaluating network communities based on ground-truth," in *Proc. SIGKDD*, 2012.

[28] J. Zhou, G. Cui, S. Hu, Z. Zhang, C. Yang, Z. Liu, L. Wang, C. Li, and M. Sun, "Graph neural networks: A review of methods and applications," *AI Open*, 2020.

[29] W. Xi, H. He, J. Gu, J. Wang, T. Yao, and Z. Liang, "A graph partitioning algorithm based on graph structure and label propagation for citation network prediction," in *Proc. KSEM*, 2023.

[30] G. Karypis and V. Kumar, "Metis—a software package for partitioning unstructured graphs, partitioning meshes and computing fill-reducing ordering of sparse matrices," 1997.

[31] Nvidia, "Nvidia v100 tensor core gpu," 2020. [Online]. Available: https://images.nvidia.com/content/technologies/volta/pdf/tesla-volta-v100-datasheet-letter-fnl-web.pdf

[32] M. O'Connor, N. Chatterjee, D. Lee, J. Wilson, A. Agrawal, S. Keckler, and W. Dally, "Fine-grained dram: Energy-efficient dram for extreme bandwidth systems," in *Proc. MICRO*, 2017.

[33] E. Kaur, "Design and analysis of d flip flop using different technologies," *International Journal of Innovative Research in Computer and Communication Engineering*, 08 2015.

[34] M. Ghasemzadeh, S. Najafibisfar, and A. Amini, "Ultra low-power, high-speed digital comparator," 2018.

[35] S. Singhal and A. Mehra, "A reduced power mux in 16nm cmos technology," in *2018 8th International Conference on Cloud Computing, Data Science Engineering (Confluence)*, 2018.

[36] Y. Wang, X. Liang, S. Niu, C. Zhang, F. Lyu, and Y. Luo, "Fdm: Fused double-multiply design for low-latency and area- and power-efficient implementation," *IEEE Transactions on Circuits and Systems II: Express Briefs*, 2024.

[37] Google and SkyWater Technology, "SkyWater Open Source PDK," 2020. [Online]. Available: https://github.com/google/skywater-pdk

[38] A. Stillmaker and B. Baas, "Scaling equations for the accurate prediction of CMOS device performance from 180 nm to 7 nm," *Integration, the VLSI Journal*, 2017.

[39] S. Lee *et al.*, "Hardware architecture and software stack for pim based on commercial dram technology: Industrial product," in *Proc. ISCA*, 2021.

[40] X. Xie, Z. Liang, P. Gu, A. Basak, L. Deng, L. Liang, X. Hu, and Y. Xie, "Spacea: Sparse matrix vector multiplication on processing-in-memory accelerator," in *Proc. HPCA*, 2021.

[41] C. Giannoula, I. Fernandez, J. G. Luna, N. Koziris, G. Goumas, and O. Mutlu, "Sparsep: Towards efficient sparse matrix vector multiplication on real processing-in-memory architectures," *POMACS*, 2022.

[42] C. Giannoula, I. Fernandez, J. Gómez-Luna, N. Koziris, G. Goumas, and O. Mutlu, "Towards efficient sparse matrix vector multiplication on real processing-in-memory architectures," in *Proc. SIGMETRICS*, 2022.

[43] Z. Zhang, H. Wang, S. Han, and W. J. Dally, "Sparch: Efficient architecture for sparse matrix multiplication," in *Proc. HPCA*, 2020.

[44] D. Baek, S. Hwang, T. Heo, D. Kim, and J. Huh, "Innersp: A memory efficient sparse matrix multiplication accelerator with locality-aware inner product processing," in *Proc. PACT*, 2021.

[45] N. Srivastava, H. Jin, J. Liu, D. Albonesi, and Z. Zhang, "Matraptor: A sparse-sparse matrix multiplication accelerator based on row-wise product," in *Proc. MICRO*, 2020.

[46] S. Liang, Y. Wang, C. Liu, L. He, H. LI, D. Xu, and X. Li, "Engn: A high-throughput and energy-efficient accelerator for large graph neural networks," *IEEE Transactions on Computers*, 2021.

[47] J. Li, A. Louri, A. Karanth, and R. Bunescu, "Gcnax: A flexible and energy-efficient accelerator for graph convolutional neural networks," in *Proc. HPCA*, 2021.

[48] L. Song, Y. Chi, A. Sohrabizadeh, Y.-k. Choi, J. Lau, and J. Cong, "Sextans: A streaming accelerator for general-purpose sparse-matrix dense-matrix multiplication," in *Proc. FPGA*, 2022.

[49] L. Ke *et al.*, "Recnmp: Accelerating personalized recommendation with near-memory processing," in *Proc. ISCA*, 2020.