Journal of
Low Power Electronics and Applications

MDPI

Article

LUPIN: A LUT-Based DRAM-PIM Accelerator for Feature Aggregation in Graph Neural Networks

Sanaz Kazemi Abharian [1], Sathwika Bavikadi [2] and Sai Manoj Pudukotai Dinakarrao [1,*]

[1] Department of Electrical and Computer Engineering, George Mason University, Fairfax, VA 22030, USA; skazemia@gmu.edu
[2] Department of Computer Engineering, Rochester Institute of Technology, Rochester, NY 14623, USA; sxbeec@rit.edu
* Correspondence: spudukot@gmu.edu

Abstract

Processing-in-memory (PIM) architectures have emerged as promising alternatives to conventional von Neumann architectures for accelerating memory-bound applications, such as Graph Neural Networks (GNNs), by reducing data movement between memory and processing units, thereby improving energy efficiency and reducing computational latency. Feature aggregation is a fundamental operation in GNNs; however, the sparsity of graph data leads to poor spatial and temporal locality. This results in dynamic and irregular memory access patterns, ultimately limiting performance and increasing latency. To address this issue, we propose LUPIN, a look-up table (LUT)-based PIM architecture to accelerate GNNs. In this design, logic-based processing is replaced with precalculated results stored in LUTs to perform the required computations within the DRAM memory platform. The proposed LUPIN architecture achieves enhanced performance and markedly improved energy efficiency compared to conventional bitwise-parallel PIM designs. Moreover, it avoids the fabrication challenges associated with integrating logic circuits into memory. It leverages the extensive internal parallelism of DRAM to accelerate the execution of GNN models while maintaining high energy efficiency. Our results demonstrate that LUPIN achieves geometric-mean energy improvements of 32× and 51× on GCN and GIN, respectively, over the CPU and 15× and 19.5×, respectively, over the GPU. Moreover, LUPIN reduces latency by 35.8× and 25.7× on GCN and GIN, respectively, compared with the CPU and 29.6× and 20.5×, respectively, compared with the GPU. The proposed LUPIN architecture achieves nearly 10× higher energy efficiency and 5× speedup compared with the state-of-the-art PIM architectures.

Keywords: processing-in-memory; DRAM; GCN; GIN; partitioning; CSC; GNNs; feature aggregation

Check for updates

Academic Editor: Mehdi Kamal

Received: 28 August 2026
Revised: 16 October 2026
Accepted: 26 October 2026
Published: 2 November 2026

Copyright: © 2026 by the authors. Licensee MDPI, Basel, Switzerland. This article is an open access article distributed under the terms and conditions of the Creative Commons Attribution (CC BY) license.

1. Introduction

Graph Neural Networks (GNNs) have emerged as a dominant paradigm in machine learning for modeling relational dependencies in non-Euclidean distance-based problems and graph-structured data, achieving state-of-the-art performance in node classification, edge prediction tasks [1,2], and graph representation learning [3]. Their effectiveness has led to widespread adoption in a range of application domains, including drug discovery [4], hardware verification [5], and recommendation systems [6].

GNNs are typically structured in multiple layers, with each layer comprising two primary operations: feature aggregation and feature transformation, also known as combi-

nation [7]. In the aggregation phase, a permutation-invariant operator (e.g., mean pooling) collects the feature vectors from a node's neighbors. The resulting aggregated representation is passed to the combination phase, where a small neural network, such as a multilayer perceptron, is used to transform the aggregated representations into lower-dimensional feature vectors, which serve as inputs to the subsequent layers.

From a computational perspective, these two phases exhibit very different characteristics. The feature transformation step requires dense–dense matrix multiplication (GEMM); therefore, it is compute-intensive. Hardware accelerators such as application-specific integrated circuits (ASICs) [8], graphics processing units (GPUs) [1], and field-programmable gate arrays (FPGAs) [9] are well-suited for accelerating this stage. In contrast, the feature aggregation phase, which corresponds to sparse–dense matrix multiplication (SpDMM), is the dominant operation in GNN models. The aggregation phase produces irregular and data-dependent memory accesses, causing bottlenecks during graph processing. This leads to increased latency and high off-chip memory traffic. One promising solution to mitigate the data movement overhead is the processing-in-memory (PIM) compute paradigm [10,11]. PIM is a computing paradigm that integrates computational capabilities directly within or near memory, enabling data to be processed close to where data resides. By reducing data transfers between processors and memory, PIM architectures significantly mitigate the data movement bottleneck. Consequently, PIM can improve both performance and energy efficiency for data-intensive workloads [12].

Recently, numerous PIM architectures have been proposed that can perform the feature aggregation step of GNNs on DRAM, SRAM [13–17] or the non-volatile STT-MRAM and SOT-MRAM memory platforms. These architectures leverage a bitline-based processing way where logic operations are performed on each bit line. In these designs, the degree of parallelism is limited. Another class of PIM architectures employs analog crossbar arrays [18]. These architectures offer excellent energy efficiency and parallelism for dense and regular data structures, particularly in convolutional neural networks (CNNs). However, the sparsity and irregularity of feature aggregation make this type of memory inefficient for accelerating GNNs.

While existing PIM architectures employ bitline-based processing components, a recent approach to PIM computation uses look-up tables (LUTs), where logic-based computations are replaced by memory look-up operations. These PIM architectures achieve superior energy efficiency and enhanced functional flexibility. Prior LUT-based PIM architectures, including LAcc [19], Pluto [20], DLUX [21], BFree [22], and pPIM [23] have demonstrated success in accelerating CNN workloads using LUTs. However, the application of LUT-based PIM to GNNs remains limited.

Unlike CNNs, GNNs exhibit irregular memory access patterns and topology-dependent computations, making existing LUT-based PIM architectures unsuitable for direct adoption. A key limitation arises from the sparse nature of graph data, which is not efficiently exploited by LUT-based PIM designs developed for CNN workloads. Consequently, these architectures may incur substantial storage redundancy by storing and processing a large number of zero-valued elements. This limitation underscores the need for developing LUT-based architectures tailored to the specific computational characteristics of GNNs. In this context, a LUT-based PIM architecture designed specifically for GNNs presents a compelling solution for accelerating feature aggregation. Leveraging LUTs in such an architecture provides greater operational flexibility with minimal overhead compared with traditional logic-based computation approaches. To the best of our knowledge, this work is the first to introduce a LUT-based PIM architecture tailored for accelerating the feature aggregation phase of GNNs.

In this work, we present LUPIN, a LUT-based PIM architecture for GNNs integrated into a DRAM device to accelerate feature aggregation. The architecture comprises multiple PIM clusters, each equipped with programmable LUTs that efficiently execute the operations central to feature aggregation. Unlike existing studies, the edge data of GNNs is leveraged as control signals rather than computational operands, enabling more efficient dataflow control. To enhance data locality and parallelism while reducing inter-bank communication, we employ vertex partitioning to map graph data onto individual DRAM banks. The proposed design enables in-situ execution of the feature aggregation step within DRAM, achieving high throughput and energy efficiency under stringent area and power constraints of memory systems. The LUPIN architecture is a programmable and reconfigurable LUT-based DRAM-PIM architecture that supports in-situ reconfiguration to implement diverse feature aggregation operators. This design enables LUPIN to adapt its computation to different GNN models and aggregation functions, providing flexibility to support multiple inference workloads.

The primary contributions of this paper are summarized below:

- We propose a reconfigurable LUT-based PIM architecture, LUPIN, that implements feature aggregation in GNNs in situ during inference. Our proposed LUT-based architecture is able to implement different feature aggregation functions in various GNNs.
- The LUPIN design leverages edge data as control signals to manage dataflow within the PIM design. This encoding mechanism effectively incorporates the graph structure into the computation and enables a simplified control scheme.
- We adopt a partition-centric method that maps the graph onto DRAM banks. By deploying partition-centric mapping, each partitioned subgraph can be processed independently and in parallel across the DRAM banks. Consequently, inter-bank communication overhead is significantly reduced.

We assess the effectiveness of the proposed LUPIN architecture in accelerating quantized GNNs, specifically GCN and GIN, across multiple benchmark datasets for inference applications. The results demonstrate that the proposed LUPIN architecture achieves about 10× higher energy efficiency and 5× speedup compared to the state-of-the-art GNN PIM architectures.

The rest of the paper is organized as follows: Section 2 introduces the fundamental concepts and provides an overview of related work. Section 3 presents the proposed LUPIN architecture. Section 4 describes the mapping of graph data into DRAM banks. Section 5 details the data flow within the PIM architecture. Section 6 discusses various technical aspects of the circuit-level implementation of the PIM cluster. Section 7 presents the performance evaluation of the PIM architecture, along with synthesis and layout data of the design. Finally, Section 8 concludes the paper.

2. Background and Related Work

In this section, we explain the operation of GNNs, focusing on the GCN and GIN models. We then review related PIM architectures for accelerating the feature aggregation phase.

2.1. Graph Neural Networks

GNNs are a class of neural networks designed to operate on graph-structured data, represented as $G = (N, E)$, where N denotes the set of nodes and E represents the edge between nodes. Message passing is the fundamental operation in GNN models, involving two phases, feature aggregation and feature transformation, as illustrated in Figure 1. The input graph consists of a set of node feature embeddings and an adjacency matrix (A) that represents the connection between nodes.

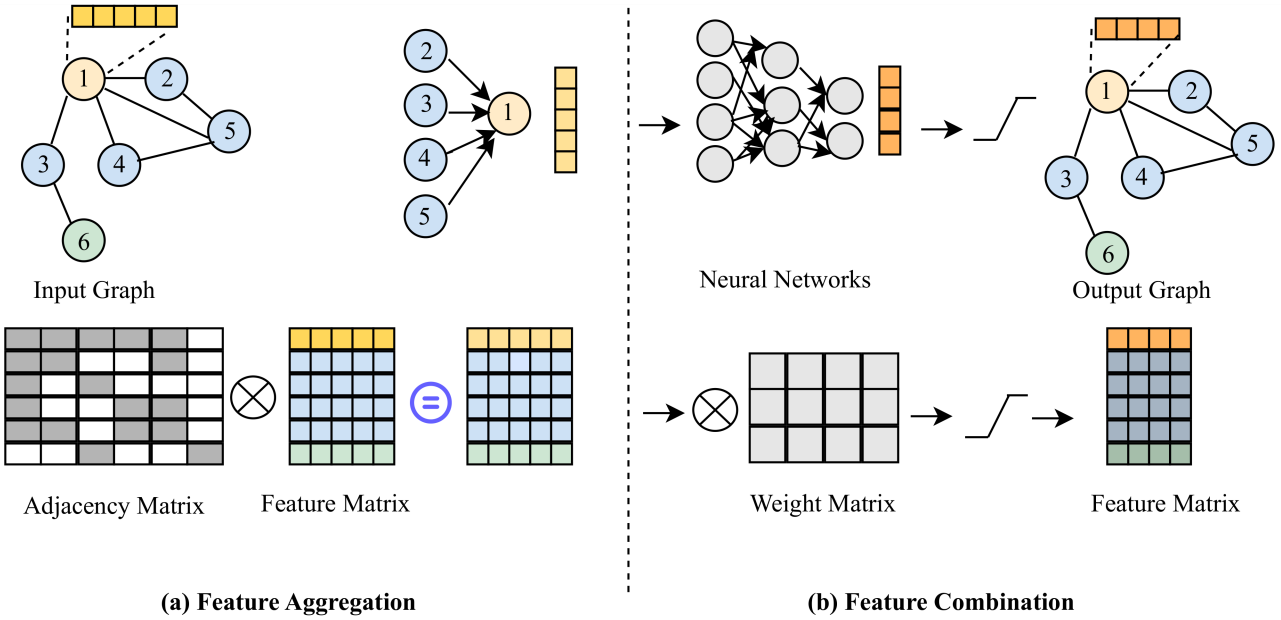


Figure 1. Illustration of the two primary operations in a message-passing layer of a GNN model: (**a**) feature aggregation and (**b**) feature transformation.

In the aggregation step, each node aggregates its neighbors' features to learn the graph's topological structure through an operator (e.g., average). In the transformation phase, the aggregated feature vector is processed through a learnable function, such as a small neural network or a linear transformation to produce a new representation of a lower dimensionality. Moreover, aggregation and combination operators differ slightly across various GNN models. Graph Convolutional Networks (GCNs) [24] extend the concept of convolution to graph-structured data by updating each node's representation using the normalized mean function from its neighboring nodes, followed by an MLP for the transformation step, as shown in Equation (1).

$$X^{l+1} = \sigma\left(\hat{A}X^{l}W\right) \tag{1}$$

In this equation, X^{l} denotes the input feature matrix, while X^{l+1} is the output feature matrix after the aggregation phase, W is the weight matrix, and $\sigma(.)$ is the activation function. The Graph Isomorphism Network (GIN) [25] is a message-passing GNN architecture designed to achieve maximal discriminative power. GIN employs a sum-based aggregation function followed by a multilayer perceptron according to Equation (2).

$$h_v^{(l+1)} = \mathrm{MLP}^{(l)}\left((1+\epsilon)\,h_v^{(l)} + \sum_{u \in \mathcal{N}(v)} h_u^{(l)}\right) \tag{2}$$

In Equation (1), the adjacency matrix (A) encodes graph connectivity and determines which neighboring node features contribute to the aggregation through the matrix multiplication $\hat{A}X^{l}$. Moreover, $\mathcal{N}(v)$ denotes the set of neighboring nodes of node v, as defined by the adjacency matrix A in Equation (2). The adjacency matrix employed in the aggregation phase is highly sparse, with the majority of its entries being zero, resulting in substantial redundant computation and increasing energy consumption in hardware implementations.

Although sparse matrix compression techniques such as Compressed Sparse Column (CSC) [26] and Compressed Sparse Row (CSR) [27] are well-suited for this type of computation, they are not considered in the current computational approach. CSR stores non-zero elements row-wise using three arrays: one for non-zero values, one for column indices, and one for row pointers. Conversely, CSC organizes the non-zero elements column by column, with analogous arrays for values, row indices, and column pointers.

2.2. Related Work

In this subsection, we briefly review prior work on DRAM-based PIM architectures for accelerating GNN workloads.

DRAM is widely used in PIM architectures due to its high storage density, low cost per bit, large capacity, and energy efficiency [28]. PIM architectures reduce the data movement that leads to significant power consumption and latency bottlenecks by embedding processing units within or close to where data resides. In DRAM-based PIM architectures, processing units can be placed at different levels of the memory hierarchy, including near-data processing (NDP), near-bank processing (NBP), and near/in-array processing (NIP) [29,30]. However, the placement of processing units within the DRAM hierarchy introduces a trade-off between data movement efficiency and computational capabilities.

Other designs place computation deeper in the DRAM hierarchy. PARAG [31] combines near-bank and near-array processing by inserting processing elements between the bit-line sense amplifiers and the column decoder of each DRAM bank to execute GCN operations. HighP [32] adopts a DIMM-based near-bank architecture for sparse matrix multiplication and integrates lightweight computation and set-operation units within DRAM banks. It further exploits in-DRAM AND/OR operations through triple-row activation to accelerate sparse computation.

Although these architectures differ in the location of computation and the organization of their processing units, their computation is implemented using conventional arithmetic, MAC, CAM, or bitwise-processing components. As a result, the functionality of the PIM architecture is closely associated with the operations supported by the integrated compute units. Among the GNN-oriented PIM architectures reviewed above, none employ LUT-based processing units to implement feature aggregation. In contrast, LUPIN introduces reconfigurable LUT-based compute units within DRAM, where arithmetic operations required by different aggregation functions are performed through accesses to precomputed results rather than dedicated arithmetic logic. This enables the same computational substrate to support different aggregation operations used by GNN models such as GCN and GIN. To the best of our knowledge, this work is the first to propose a LUT-based PIM architecture for GNN workloads.

3. Proposed LUPIN PIM Architecture

3.1. LUPIN Design Overview

The proposed LUPIN architecture is designed to execute the feature aggregation phase in GNN models. Figure 2 presents a hierarchical representation of the architecture, including (a) the vertex-centric method for partitioning graph data across multiple DRAM banks, (b) the arrangement of PIM clusters embedded between the DRAM subarrays within each bank, (c) the internal structure of a PIM cluster as a programmable processing element (PE), and (d) the microarchitecture of individual PIM cores.

Each PIM cluster functions as an independent processing element that is programmed to perform a variety of arithmetic operations, such as multiplication, addition, and bitwise logic, which are necessary for implementing diverse feature aggregation strategies in GNNs. These strategies differ across models: GCN employs a normalized mean operator and GIN uses a sum-based function. By reconfiguring LUT entries, the proposed LUPIN architecture can flexibly support these aggregation operators within the same hardware substrate, ensuring compatibility with multiple GNN frameworks.

The PIM clusters are organized in horizontal arrays within the DRAM bank, with each row positioned between two memory subarrays. A PIM cluster accesses the segment of the subarray directly above it to read and write data operands. All PIM clusters within a bank operate concurrently and independently, providing high parallel processing throughput.

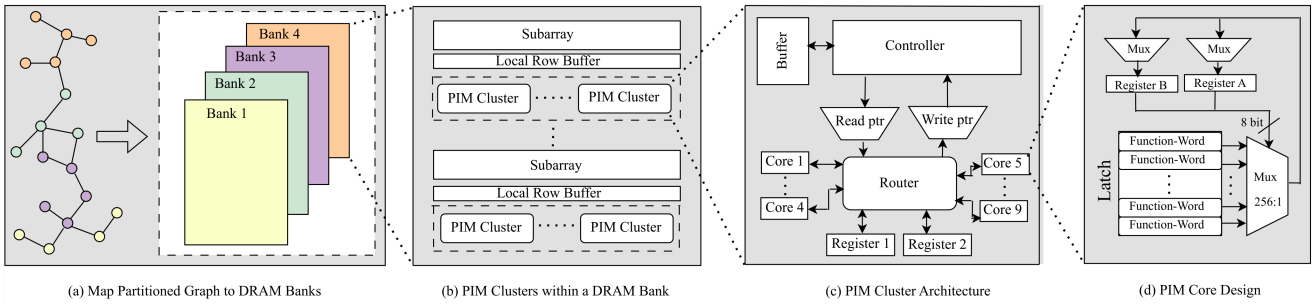


Figure 2. Hierarchical overview of the proposed LUPIN design, illustrating the mapping graph partitioning to DRAM banks, the organization of PIM clusters within a DRAM bank, and the microarchitecture of a LUT core within a PIM cluster, respectively from left to right.

3.2. PIM Cluster Architecture

A PIM cluster is a processing element that integrates the functions of its internal LUT cores to execute required feature aggregation operations. In addition to the LUT cores, each cluster incorporates a pair of 8-bit registers that facilitate the routing of intermediate data during the execution of the aggregation function. Moreover, a router interconnects the cores within a cluster to enable data communication and parallelism. Thus, it enables simultaneous communication between LUT cores, 8-bit registers, and read and write pointers. The routing strategy enables data exchanges between cores to be divided into upper and lower 4-bit segments, each of which can be transmitted independently. This adoption of 4-bit segmented routing is motivated by the need to balance area and timing. Segmenting 8-bit operands into upper and lower 4-bit halves reduces interconnect complexity and shortens critical paths, all of which are crucial within the tight layout constraints of DRAM banks. This functionality is realized using 10-to-1, 4-bit multiplexers that collectively constitute a SPIN-based interconnection fabric.

3.3. PIM Core Architecture

The PIM core is tailored to perform any logic and arithmetic operation by looking up pre-calculated results instead of fixed logic circuits. Moreover, 8-bit precision for GNN inference has recently attracted significant interest because it substantially reduces computational overhead, which is used in the majority of graph data in real-world applications. Thus, we consider 8-bit integer precision in our design as the baseline.

Each PIM core consists of 256 8-bit latches that store pre-computed results of the required operations and are loaded as function words from the subarray bitlines. These latches are connected to 8:1 multiplexers to produce intended output. The output can either be sent out of the LUT core through the router or fed back into registers for another operation cycle, if needed.

Inputs are provided via a pair of registers, A and B. Thus, these registers control the selected inputs of the multiplexer and retrieve a specific 8-bit value from the eight latches that store the operation's output. Furthermore, the input operands are obtained through the PIM cluster's router into a pair of 4-bit registers, A and B, whose combined values drive the multiplexer select lines. This configuration enables the multiplexer to retrieve the corresponding 8-bit entry from the eight latch cells that represent the operation's output.

LUPIN leverages reconfigurable fixed-size LUT cores to support GNN models with varying layer and operation sizes, avoiding proportional growth in LUT capacity. Reconfiguration overhead is minimal since LUTs are programmed once per operation type and remain static during inference, the LUT contents remain fixed, and runtime execution only adjusts data routing and updates index values that select LUT entries. Consequently, LUT reconfiguration is performed infrequently, with its overhead amortized across a large num-

ber of operations. As a result, it does not lie on the critical path, enabling scalable execution without additional LUT capacity or reconfiguration overhead.

3.4. Feature Aggregation Implementation in LUPIN

In this subsection, we describe how LUPIN configures its PIM clusters to efficiently execute feature aggregation. The baseline design focuses on the GCN model. We then illustrate how the LUT cores can be programmed to implement the feature aggregation function of the GIN model.

Each PIM cluster contains nine LUT cores, which are programmed to execute the operations required for the feature aggregation phase of the GCN model. The nine-LUT-core formation of the PIM cluster allows the implementation of one complete 8-bit feature aggregation operation by programming four cores to perform multiplications and five cores to perform additions. In GCN, each feature aggregation node acts as a destination node, while its neighbors serve as source nodes. Figure 3 presents the dataflow of a destination node with a source node. To implement the 8-bit multiplication operation using LUT cores, the input operands are divided into upper and lower 4-bit segments. During the first clock cycle ($t = 1$), four LUT cores (LUT 1–LUT 4) operate in parallel to compute the partial products of these segments. In the subsequent clock cycles ($t = 2$ and $t = 3$), the intermediate results are routed to LUT cores configured for addition to combine the partial products. During the fourth clock cycle ($t = 4$), LUT 1 and LUT 2 are reused to perform the final addition operations, demonstrating the reconfigurability of the proposed architecture. Figure 4 presents the dataflow model that implements the sum aggregation used for feature aggregation in GIN. In the first clock cycle ($t = 1$), four LUT cores perform addition operations on the upper and lower 4-bit segments of the input features. The intermediate results are subsequently combined through multiple stages of LUT-based additions during the second and third clock cycles ($t = 2$ and $t = 3$). Finally, LUT 9 performs the last addition operation in the fourth clock cycle ($t = 4$). These figures demonstrate how the proposed LUPIN architecture supports different feature aggregation operations through the reconfiguration and reuse of LUT cores across successive clock cycles.

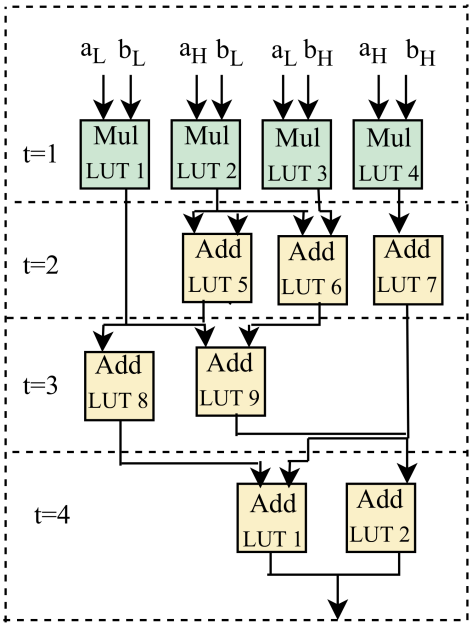


Figure 3. GCN feature aggregation dataflow in LUPIN, illustrating how an 8-bit multiply-accumulate operation between a destination node and a source node is mapped onto multiple LUT cores.

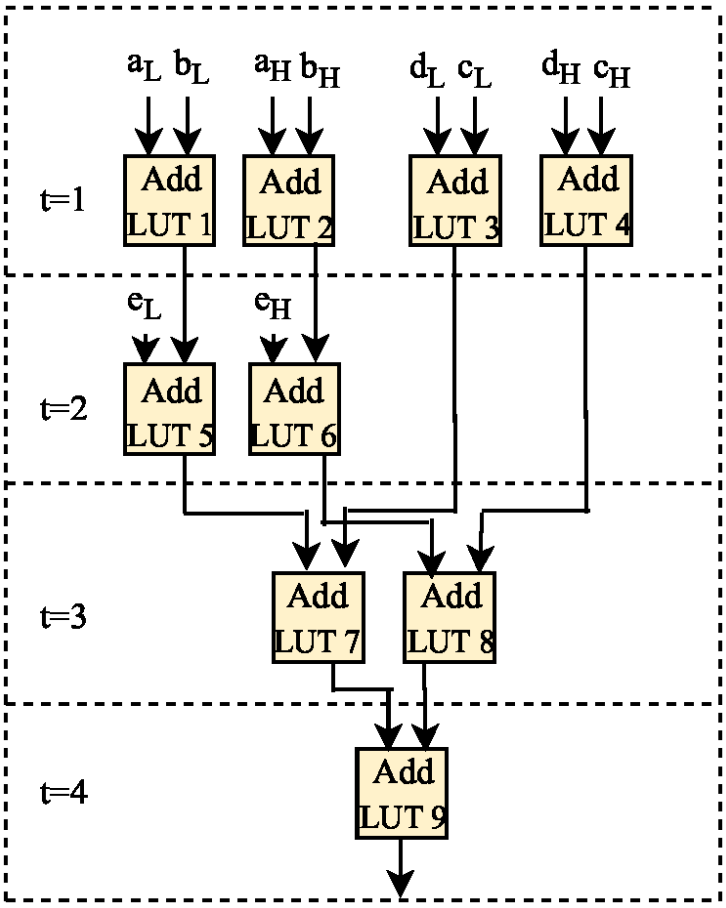


Figure 4. Dataflow of GIN feature aggregation in LUPIN, illustrating LUT-based sum aggregation for accumulating neighbor features of a destination node.

4. Partition-Based Graph Mapping

Real-world input graphs used in GNNs are typically massive, comprising tens of millions of nodes and connections [33]. To enable scalable and parallel execution, we adopt a partition-based graph mapping strategy. This pre-processing step is essential to enabling parallel processing, improving data locality, balancing workload across DRAM banks, and minimizing inter-bank communication overhead.

In this work, we apply a vertex-centric partitioning approach, where the set of nodes in the input graph is divided into K disjoint subsets, with K corresponding to the number of DRAM banks in the memory. We choose a vertex-centric scheme since computation in GNN feature aggregation is naturally organized around destination vertices. For each destination vertex, the architecture must gather and combine the features of its neighboring source vertices. Therefore, assigning vertices to banks provides a direct mapping between the feature aggregation process and PIM cluster parallelism. In contrast, edge-centric partitioning distributes edges rather than destination-node computations, which increases coordination overhead and complicates accumulation of partial sums across banks.

Moreover, the partitioning process is performed once as an offline preprocessing step, after which the data layout remains fixed throughout inference.

Finally, during uncoarsening, the graph is incrementally expanded back to its original size while refining the partitions to enhance their quality. This hierarchical strategy allows METIS to produce high-quality partitions with minimal edge cuts, making it well-suited for our parallel DRAM-based architecture. METIS is employed to reduce, rather than eliminate, inter-partition edges. For an edge whose source and destination nodes are assigned to

different banks, the source feature is accessed from its home bank and transferred to the bank processing the destination node. The original source feature remains unchanged at its home location. By reducing the number of cross-partition edges, METIS decreases the frequency of such inter-bank feature transfers and consequently reduces communication overhead.

To generate these partitions, we use METIS [34], which aims to balance partition sizes while minimizing edge cuts, i.e., graph edges whose two endpoints are assigned to different partitions. For the actual partitioning algorithm, we utilize METIS [34], a widely adopted tool in GNN frameworks for graph partitioning [35]. METIS employs a multilevel recursive bisection scheme that includes three key phases: graph coarsening, initial partitioning, and uncoarsening with refinement. During coarsening, the graph is progressively collapsed to form a series of smaller graphs. An initial partitioning is then performed on the coarsest graph. Finally, during uncoarsening, the graph is incrementally expanded back to its original size while refining the partitions to enhance their quality. This hierarchical strategy allows METIS to produce high-quality partitions with minimal edge cuts, making it well-suited for our parallel DRAM-based architecture.

5. Data Flow of Proposed LUPIN

We adopt a dataflow mechanism in the LUPIN architecture to control graph node data. To this end, we repurpose the adjacency matrix as a set of control signals that support this mechanism.

In the aggregation phase, each node can act as either a destination or a source node. As a destination node, it aggregates features from its neighboring source nodes. Therefore, as a source node, the node's feature vector must remain constant until all its neighboring destination nodes complete aggregation, whereas as a destination node, its feature value updates during the aggregation process. Consequently, in our design, the destination node remains in the PIM cluster to receive data from its source nodes and perform feature aggregation.

During feature aggregation, the input node features and the generated output features are maintained separately. Therefore, updating a destination node's output feature does not overwrite its original input feature, which may still be required as a source feature by other neighboring nodes. After aggregation for the current GNN layer is completed, the generated output features become the input features for the subsequent layer.

Data Flow Controller

In the proposed LUPIN design, the data-flow controller identifies the source nodes of each destination node solely based on the structural information of the graph, independent of the node features. To enable this, we employ the graph adjacency matrix as the control signal for the controller. The adjacency matrix is represented in the CSC format to sequentially access all relevant neighbors.

The controller converts the adjacency matrix into CSC format, which is suitable for our design since only one destination vertex is processed at a time. Moreover, the aggregation phase is destination-centric and CSC allows direct, sequential access to all relevant neighbors, enabling efficient address generation and dataflow control in the LUT-based PIM cluster. Figure 5 illustrates a representative example of translating an adjacency matrix into the CSC structure and extracting the source nodes for each destination node during feature aggregation. Array I represents the source nodes and the array P provides the starting indices for each destination node. This encoding allows the architecture to sequentially access only the relevant source vertices for a target destination vertex, thereby minimizing unnecessary memory access.

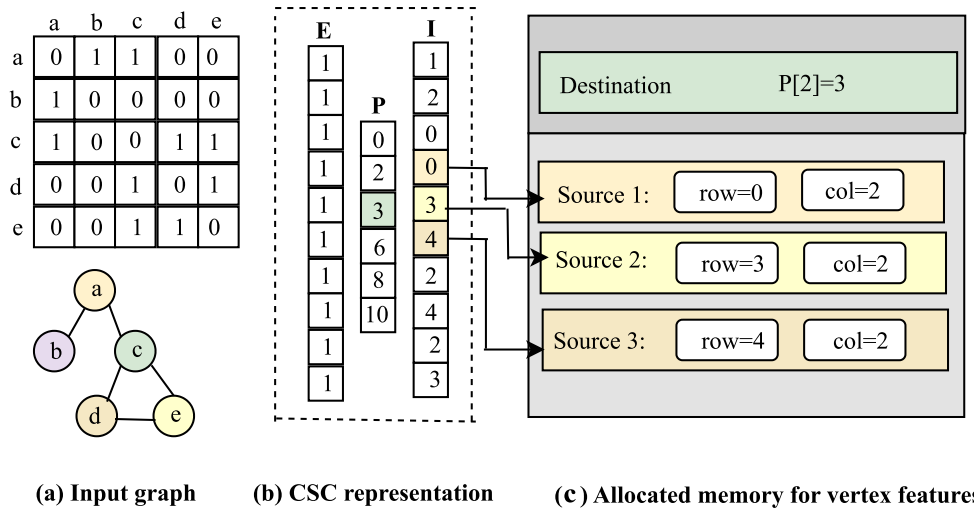


Figure 5. Depiction of employing an adjacency matrix to determine the source vertices corresponding to a destination node in memory (**a**) input graph, (**b**) CSC representation of the adjacency matrix, and (**c**) addressing source vertices of a destination vertex based on the CSC format.

Each PIM cluster is paired with a dedicated controller. The controller consists of a control-signal generator and local register files. It also includes a simple pointer decoder that employs the CSC arrays (P and I) to identify the range of neighbors for each destination node. In addition, the local register buffers the source-node values to ensure that source features remain constant while being reused across multiple aggregation steps. Once all neighbors are processed, the destination node's feature vector is updated and written back to its original location.

The controller is independent of feature dimensionality and aggregation operator, as its operation depends solely on the structural encoding of the graph. By reprogramming the LUT entries, the same hardware supports both GCN with normalized mean aggregation and GIN with sum aggregation. If the adjacency matrix is modified, reconfiguration is achieved by regenerating the CSC representation offline.

6. Experimental Evaluation

In this section, we evaluate the proposed LUPIN architecture in terms of latency, energy consumption, throughput, and area to implement feature aggregation phase for GNN models. Additionally, we compare LUPIN with state-of-the-art existing PIM architectures and implementing on CPU and GPU.

Workloads: We evaluate two types of GNN models, GCN and GIN, using four commonly employed benchmark datasets: Cora, Citeseer, PubMed, and Reddit, summarized in Table 1. These datasets are selected to capture diverse characteristics, including variations in the number of nodes, graph density, and feature dimensionality, which in turn highlight either the aggregation phase, the combination phase, or provide a balanced contribution to the overall latency. The proposed LUT-based architecture is not inherently limited to 8-bit precision and can be configured to support different numerical precisions by appropriately modifying the LUT design. However, since the evaluated GNN workloads employ 8-bit quantized features, we adopt LUT cores designed for 8-bit fixed-point integer arithmetic to perform feature aggregation. The LUPIN PIM architecture is simulated and verified using a Verilog HDL model, as summarized in Table 2. Additionally, we compare LUPIN with existing state-of-the-art PIM architectures summarized in Table 3, as well as CPU and GPU implementations.

Graph Partitioning: To evaluate the proposed LUPIN architecture, we consider graph partitioning counts of 1, 2, 4, and 8 as a case study. Since we use a DDR3 memory system that consists of eight banks, evaluating up to eight partitions reflects the practical hardware limit. Intermediate values (2 and 4) allow us to analyze scalability trends and balance partitioning overhead against parallel throughput. This range of configurations ensures a comprehensive case study across datasets of varying sizes and densities.

Table 1. Dataset Characteristics.

Datasets	Nodes	Edges	Features	Classes	Density of A
Cora	2708	10,556	1433	7	0.144
Citeseer	3327	9104	3703	6	0.082
PubMed	19,717	88,648	500	3	0.023
Reddit	232,965	114,615,892	602	41	0.042

6.1. Comparative Assessment of LUPIN Performance Across Partitioning Configurations

Figures 6 and 7 present the normalized energy consumption of the proposed LUPIN architecture for GCN and GIN models, while Figures 8 and 9 report the corresponding latency results. These evaluations demonstrate how the LUT-based design inherently delivers energy-efficient and low-latency feature aggregation, with graph partitioning serving as a complementary mechanism to scale performance across datasets of different sizes and densities.

For energy consumption according to the Figure 6 and 7, all configurations employ the LUT-based PIM architecture, with differences arising from the number of partitions. Partitioning consistently lowers energy consumption relative to the no-partitioning baseline distributing workloads more evenly across DRAM banks and reducing inter-bank communication. The benefit is modest for small datasets such as Cora and Citeseer, but becomes substantial for larger graphs, PubMed and Reddit, where eight partitions achieve the lowest normalized energy. These results indicate that partition-centric mapping amplifies the baseline efficiency already provided by the LUT-based design.

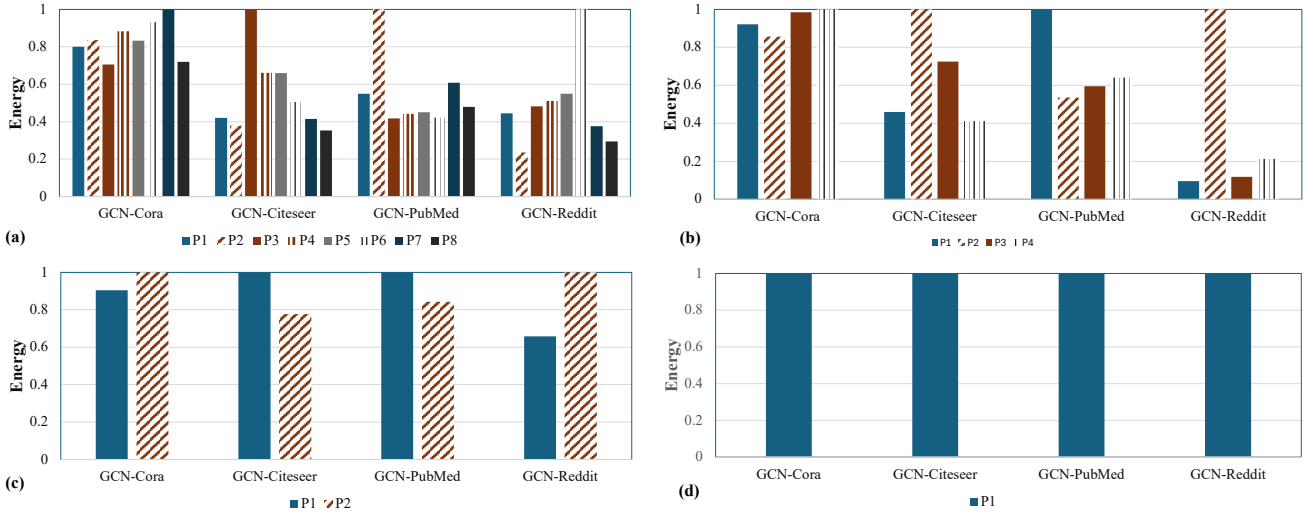


Figure 6. Normalized energy consumption of the GCN on the LUPIN architecture under different vertex partitioning configurations: (**a**) eight partitions, (**b**) four partitions, (**c**) two partitions, and (**d**) one partition.

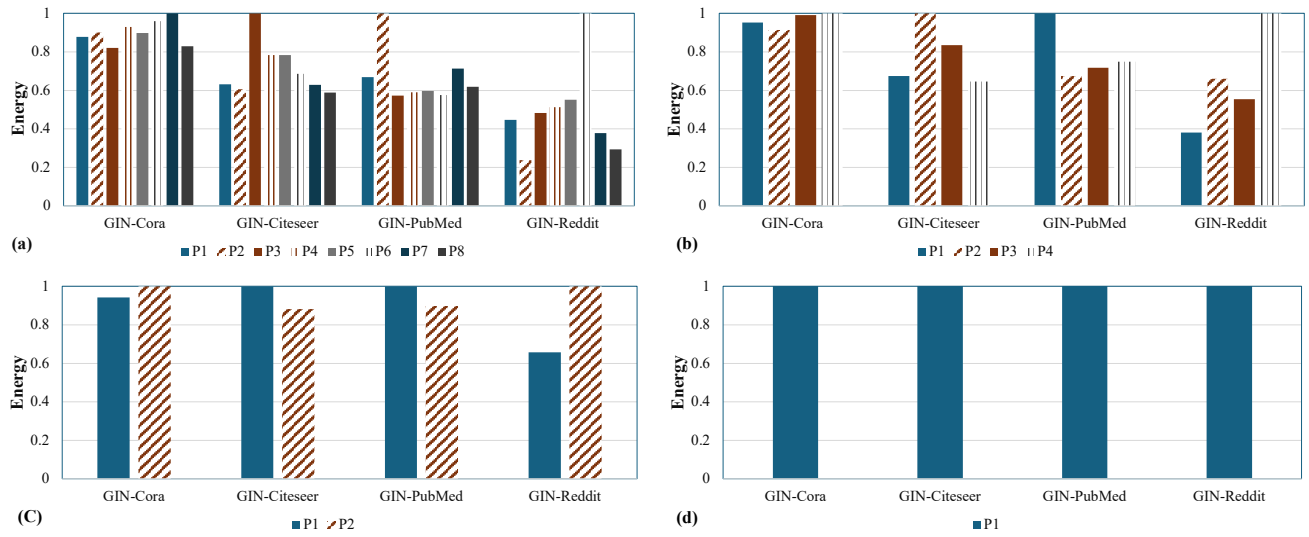


Figure 7. Normalized energy consumption of the GIN model on the LUPIN architecture under various vertex partitioning configurations: (**a**) eight partitions, (**b**) four partitions, (**c**) two partitions, and (**d**) one partition.

Figures 8 and 9 present the latency evaluation of the proposed LUPIN architecture for GCN and GIN models under different partitioning configurations. For the GCN model, partitioning yields consistent improvements across all datasets. The reduction is modest for Cora (13%) but becomes more pronounced for Citeseer (26%) and PubMed (16%). The most substantial improvement is observed for Reddit, where latency decreases by more than 33% when scaling from one to eight partitions. Similarly, the GIN model demonstrates latency reductions of 9% for Cora, 14% for Citeseer, and 12% for PubMed. Reddit again shows the largest benefit, with a reduction exceeding 34%. These results indicate that partition-centric mapping effectively accelerates inference, with larger and denser graphs exhibiting the greatest latency improvements.

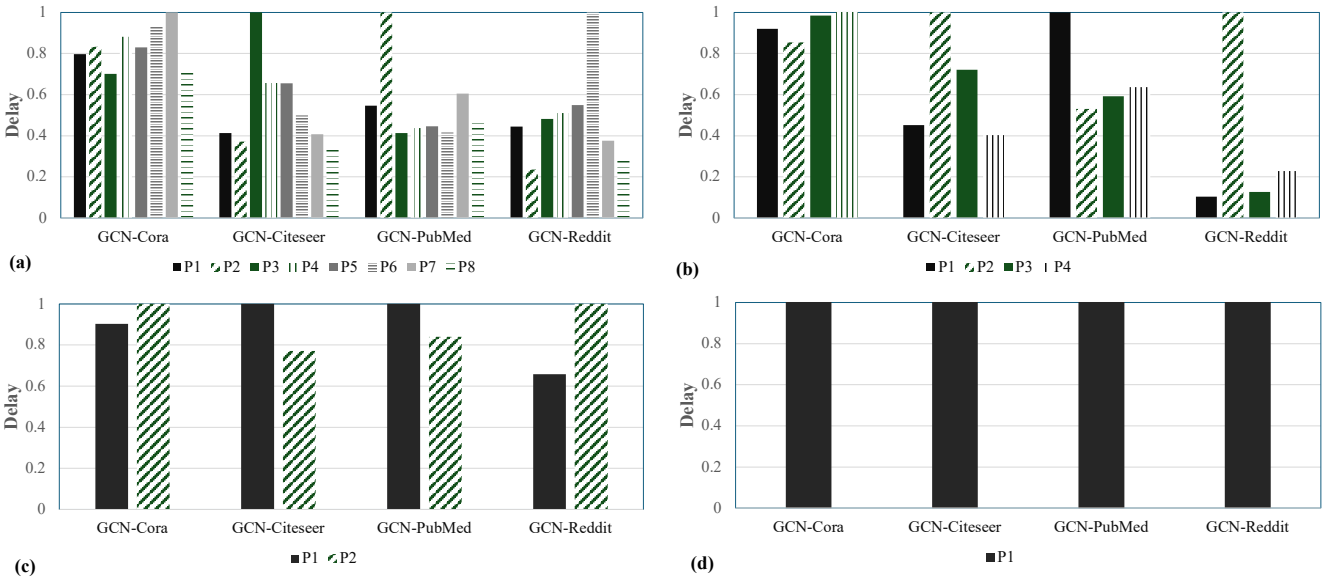


Figure 8. Normalized latency of the GCN model under different vertex partitioning configurations on the LUPIN architecture: (**a**) eight partitions, (**b**) four partitions, (**c**) two partitions, and (**d**) one partition.

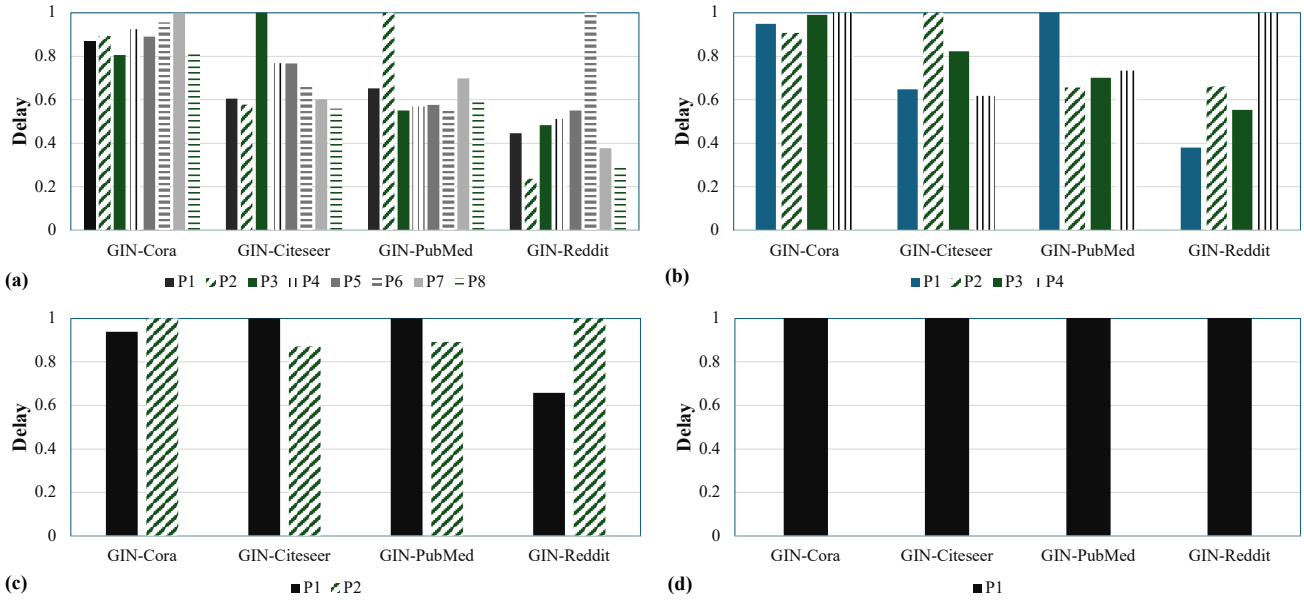


Figure 9. Normalized latency of the GIN model implemented on the LUPIN architecture under varying vertex partitioning configurations: (**a**) eight partitions, (**b**) four partitions, (**c**) two partitions, and (**d**) one partition.

Table 2. Characteristics of PIM components implemented in the 28 nm Technology Node.

Component	Power (mW)	Active Area (μm^2)
PIM Cluster	13.5	37,769.81
LUT Core	2.7	4196.64
Intra-Subarray Communication [36]	0.028 µJ/comm	N/A
Inter-Subarray Communication [37] for subarrays 1/7/15 hops away	0.09/0.12/0.17 µJ/comm	N/A

Table 3. Performance comparison of LUPIN with PARAG.

PIM Architecture	Datasets	Normalized Latency	Normalized Energy Efficiency
Proposed LUPIN Architecture	Cora, Citeseer, PubMed, Reddit	5×	10×
PARAG [31]	Reddit, Citeseer	1	1

6.2. Accuracy Comparison

A primary challenge associated with vertex partitioning during graph inference is the potential degradation of model prediction accuracy. Figure 10 illustrates the impact of different partition counts on the test accuracy of GCN and GIN across the Cora, Citeseer, PubMed, and Reddit datasets. Overall, increasing the number of partitions from 1 to 8 results in relatively small accuracy variations for most model–dataset combinations, indicating that graph partitioning generally preserves model accuracy. In particular, GCN exhibits negligible accuracy degradation on Cora and Citeseer and only a modest reduction on Reddit, while GCN-PubMed shows a more noticeable decline as the number of partitions increases. A similar trend is observed for GIN, where accuracy remains relatively stable on Cora, Citeseer, and Reddit. However, GIN-PubMed is more sensitive to partitioning, with a substantial accuracy reduction at eight partitions. Although METIS minimizes cross-partition edges while preserving graph connectivity, increasing the number of partitions may introduce slight variations in feature aggregation due to changes in computation order and

inter-bank feature transfers. These variations may propagate through successive GNN layers and affect final classification results, particularly for nodes with similar class probabilities. Consequently, minor accuracy variations may occur as the number of partitions increases, even though all graph edges are preserved.

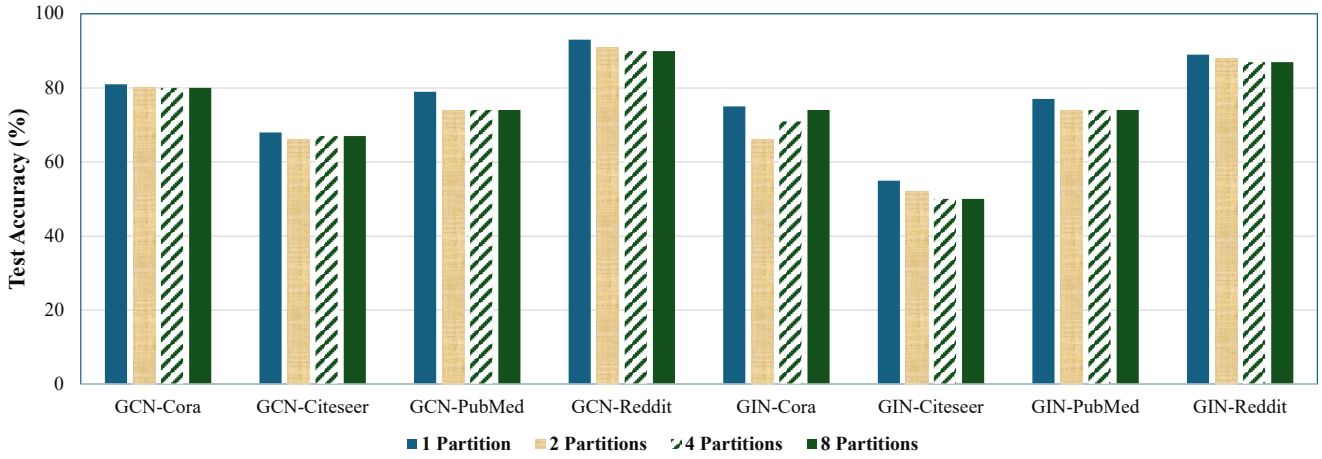


Figure 10. Average test accuracy of GCN and GIN models on Cora, Citeseer, PubMed, Reddit datasets under different partition counts (1, 2, 4, and 8).

6.3. Hardware Implementation

In this work, GNN models exclusively operate on 8-bit data. Thus, the PIM architecture employs a fixed 8-bit LUT as its fundamental computation unit. Moreover, key components of the LUT cores, including the latch arrays and 256:1 multiplexers, are implemented by Transmission Gate (TG) logic. The multiplexers are organized in multiple TG stages, with CMOS buffer insertion between stages to ensure signal integrity and prevent metastability. The power and delay for the PIM cluster and LUT core are estimated using Synopsys Design compiler with the 28 nm standard cell libraries from TSMC, and results are summarized in Table 2. The power consumption of the PIM cluster is determined by the number of LUT cores active during each clock cycle. As illustrated in Figures 3 and 4, not all nine LUT cores operate simultaneously during feature aggregation. Instead, different subsets of LUT cores are activated in successive clock cycles according to the computational dataflow. Consequently, the reported PIM cluster power consumption accounts for the activity of the LUT cores during execution rather than assuming that all nine cores are active simultaneously. Moreover, the power consumption associated with inter-bank communication is estimated using the intra-subarray and inter-subarray power consumption values reported in Table 2.

A group of PIM clusters are integrated within each DRAM bank, arranged in rows that alternate with many subarrays. Each row of PIM clusters can access data operands from the neighboring subarrays. By tuning the timing parameters within the memory controller, multiple subarrays can be activated concurrently, as demonstrated in [38]. This configuration enhances the modularity of the PIM design while maintaining minimal disruption to the original DRAM organization. Furthermore, since the memory subarrays are not utilized for LUT implementation, the operating frequency remains unaffected by the latency associated with row activation and read operations [19,20]. In addition, the inclusion of a read/write buffer enables the PIM architecture to fetch larger data blocks per access, effectively hiding data-access latency. Consequently, the next target row for operand read or write can be pre-activated while the current data row is still under processing within the PIM cluster.

Eight PIM clusters can be arranged in a row, occupying an area approximately equal to that one memory subarray based on the area of a PIM cluster in Table 2. We adopt a DRAM chip for the proposed LUPIN PIM containing 32 subarrays per memory bank [39]. Within a PIM-enabled bank, we configure 16 rows of PIM clusters placed alternately with 16 subarrays, forming a total of 128 PIM clusters distributed across the bank.

7. Comparative Performance Evaluation with State-of-the-Art PIM Architectures

We evaluate the proposed LUPIN architecture against state-of-the-art PIM design, including PARAG [31], as well as CPU- and GPU-based GNN implementations. To enable a fair comparison in terms of energy consumption and latency, we implement GCN and GIN models using the PyTorch Geometric framework [40] on both CPU and GPU platforms. The evaluation focuses on the same feature aggregation phase to ensure a consistent computational scope across all evaluated platforms. The detailed CPU and GPU configurations are summarized in Table 4. All evaluations focus on the aggregation phase of GNN inference, and energy consumption and latency are reported.

Table 4. Configuration of baselines.

Baseline	Intel Xeon CPU	NVIDIA Tesla T4 GPU
Compute units	3.2 GHz @ 8 Cores	1.59 GHz @ 2560 CUDA cores
On-chip memory	16 MB	4 MB L2 cache
Off-chip memory	68.25 GB/s DDR4	16 GB GDDR6 @ 320 GB/s

According to Table 3, the LUPIN architecture achieves approximately a 40× reduction on Citeseer and 20–40× and 30–35× on PubMed and Reddit, respectively, when compared to the CPU. In contrast, the PARAG architecture achieves a latency reduction of 4–7× on smaller datasets and up to 20–25× on Reddit relative to the CPU. In contrast, the proposed LUPIN architecture consistently delivers higher performance. Furthermore, LUPIN achieves approximately 30–75× energy reduction, whereas PARAG provides only 2–4× improvement.

Figures 11 and 12 compare the energy consumption of the feature aggregation phase across the LUPIN architecture, CPU, and GPU. Relative to CPU execution, LUPIN achieves up to 75× lower energy consumption, while providing up to 14× energy reduction compared to GPU for the GCN model. For the GIN model, LUPIN demonstrates higher energy efficiency, achieving up to 30× lower energy consumption on the PubMed, and Citeseer datasets and approximately 80× lower energy consumption on the Reddit dataset compared to GPU implementation.

Figures 13 and 14 compare the latency of the GCN and GIN models executed on CPU, GPU, and the proposed LUPIN architecture. Relative to CPU execution, LUPIN achieves up to 39× lower latency for the GCN model, while providing up to 38× lower latency compared to GPU execution. For the GIN model, LUPIN reduces inference latency by up to 40× relative to the CPU and by up to 41× relative to the GPU.

Compared to CPU execution, LUPIN achieves substantial savings across all datasets, with the relative advantage scaling with dataset size. For smaller graphs, energy reductions are moderate because computation is limited by partitioning overheads. In contrast, for larger graphs (PubMed and Reddit), LUPIN reduces energy by over an order of magnitude, underscoring the benefit of in-situ aggregation and reduced data movement. These results demonstrate that LUT-based processing is highly effective in minimizing redundant operations while leveraging DRAM parallelism to maintain energy efficiency.

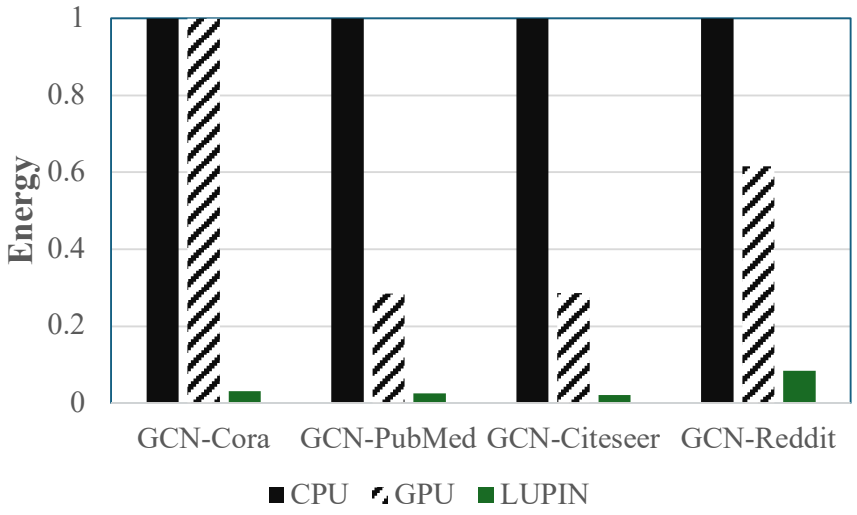


Figure 11. Normalized energy consumption of the GCN model averaged over vertex partitioning configurations (1, 2, 4, and 8) on CPU, GPU, and the LUPIN architecture.

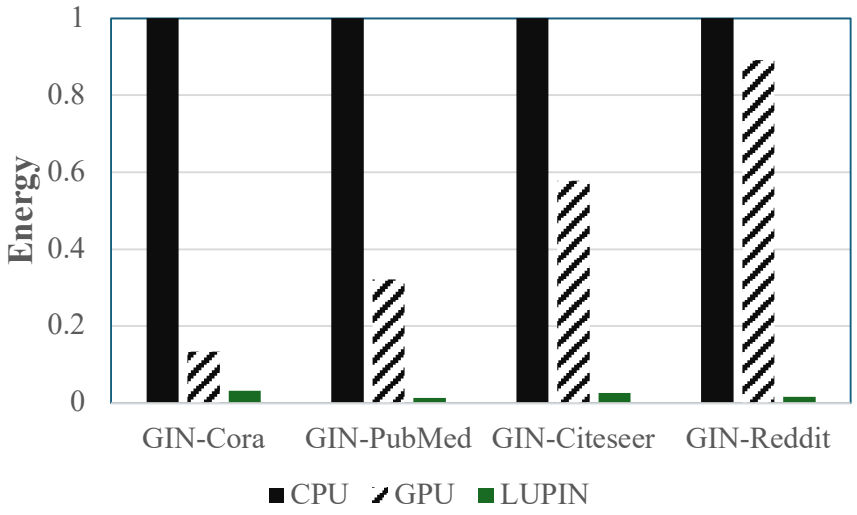


Figure 12. Energy comparison of GIN on CPU, GPU and LUPIN architecture, averaged over 1, 2, 4, and 8 partitions.

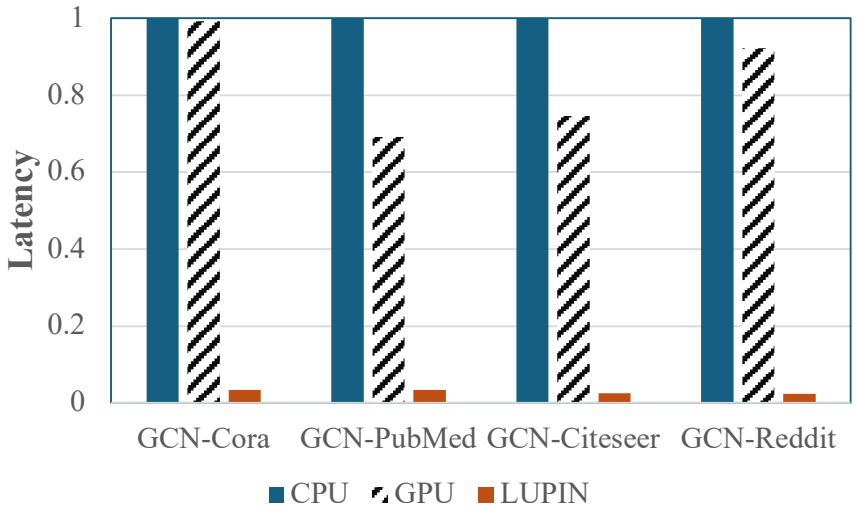


Figure 13. Latency comparison of GCN on CPU and LUPIN, averaged over 1, 2, 4, and 8 partitions.

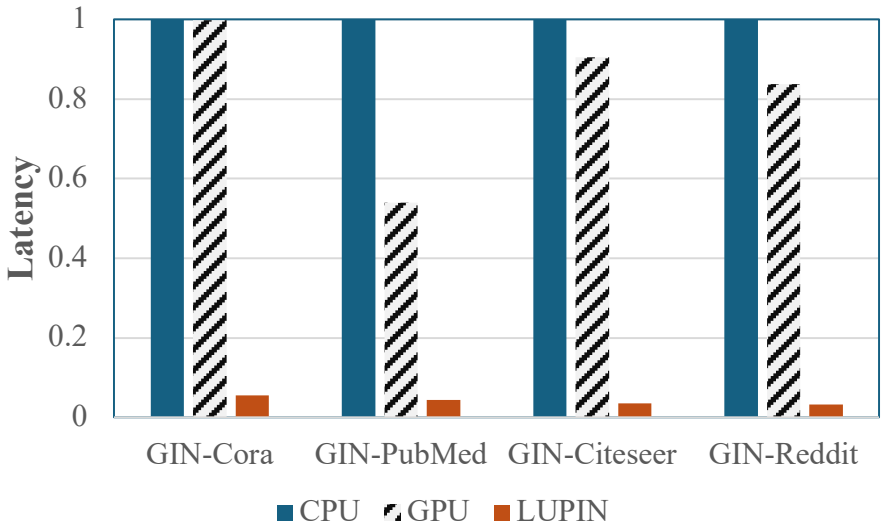


Figure 14. Latency comparison of GIN on CPU and LUPIN Architecture, averaged over 1, 2, 4, and 8 partitions.

8. Conclusions

This paper presents LUPIN, a reconfigurable LUT-based PIM architecture tailored to efficiently compute feature aggregation in GNN inference. LUPIN employs LUTs to enable in-memory computation, leveraging their functional programmability to execute arbitrary arithmetic or logical operations. This flexibility supports a wide range of feature aggregation functions used across different GNN models. To improve scalability and workload balance, we adopt vertex-centric partitioning that maps subgraphs to DRAM banks. Our results demonstrate that LUPIN exhibits varying levels of improvement on both GCN and GIN models across different datasets and significantly enhances energy efficiency and latency performance on large-scale datasets. Also, we evaluate the performance of LUPIN against state-of-the-art CPU, GPU, and PIM architectures.

Author Contributions: Conceptualization, S.K.A. and S.M.P.D.; methodology, S.K.A., S.B. and S.M.P.D.; software, S.K.A. and S.B.; validation, S.K.A. and S.B.; formal analysis, S.K.A. and S.B.; investigation, S.K.A.; resources, S.M.P.D.; data curation, S.K.A.; writing—original draft preparation, S.K.A.; writing—review and editing, S.K.A., S.B. and S.M.P.D.; visualization, S.K.A.; supervision, S.M.P.D.; project administration, S.M.P.D.; funding acquisition, S.M.P.D. All authors have read and agreed to the published version of the manuscript.

Funding: This work was supported in part by the US National Science Foundation (NSF) Grant TI-2507092. The views, opinions, and/or findings contained in this article are those of the author(s) and should not be interpreted as representing the official views or policies, either expressed or implied, of the US NSF.

Data Availability Statement: All the experimental data supporting the findings of this study are provided within the article. Additional inquiries can be directed to the corresponding author.

Conflicts of Interest: The author declares no conflicts of interest.

References

1. Ma, L.; Yang, Z.; Miao, Y.; Xue, J.; Wu, M.; Zhou, L.; Dai, Y. {NeuGraph}: Parallel deep neural network computation on large graphs. In Proceedings of the 2019 USENIX Annual Technical Conference (USENIX ATC 19), Renton, WA, USA, 10–12 July 2019; pp. 443–458.
2. Abharian, S.K.; Dinakarrao, S.M.P. GBFA: Gradual bit-flip fault attack on graph neural network accelerators. In *Proceedings of the 2026 27th International Symposium on Quality Electronic Design (ISQED)*; IEEE: New York, NY, USA, 2026; pp. 1–8.

3. Guan, F.; Zhu, T.; Zhou, W.; Choo, K.K.R. Graph neural networks: A survey on the links between privacy and security. *Artif. Intell. Rev.* **2024**, *57*, 40. [CrossRef]
4. Zeng, X.; Feng, P.K.; Li, S.J.; Lv, S.Q.; Wen, M.L.; Li, Y. GNN-DDAS: Drug discovery for identifying anti-schistosome small molecules based on graph neural network. *J. Comput. Chem.* **2024**, *45*, 2825–2834. [CrossRef] [PubMed]
5. Saravanan, R.; Kasarapu, S.; Manoj Pudukotai Dinakarrao, S. GraphFuzz: Accelerating Hardware Testing with Graph Models. *arXiv* **2024**, arXiv:2412.13374.
6. Wang, B.; Chen, J.; Li, C.; Zhou, S.; Shi, Q.; Gao, Y.; Feng, Y.; Chen, C.; Wang, C. Distributionally robust graph-based recommendation system. In Proceedings of the ACM Web Conference 2024, Singapore, 13–17 May 2024; pp. 3777–3788.
7. Etemadyrad, N.; Gao, Y.; Manoj Pudukotai Dinakarrao, S.; Zhao, L. Global explanation supervision for Graph Neural Networks. *Front. Big Data* **2024**, *7*, 1410424. [CrossRef] [PubMed]
8. Yan, M.; Deng, L.; Hu, X.; Liang, L.; Feng, Y.; Ye, X.; Zhang, Z.; Fan, D.; Xie, Y. Hygcn: A gcn accelerator with hybrid architecture. In *Proceedings of the 2020 IEEE International Symposium on High Performance Computer Architecture (HPCA)*; IEEE: New York, NY, USA, 2020; pp. 15–29.
9. Lin, Y.C.; Zhang, B.; Prasanna, V. Gcn inference acceleration using high-level synthesis. In *Proceedings of the 2021 IEEE High Performance Extreme Computing Conference (HPEC)*; IEEE: New York, NY, USA, 2021; pp. 1–6.
10. Lee, S.; Kang, S.h.; Lee, J.; Kim, H.; Lee, E.; Seo, S.; Yoon, H.; Lee, S.; Lim, K.; Shin, H.; et al. Hardware architecture and software stack for PIM based on commercial DRAM technology: Industrial product. In *Proceedings of the 2021 ACM/IEEE 48th Annual International Symposium on Computer Architecture (ISCA)*; IEEE: New York, NY, USA, 2021; pp. 43–56.
11. Mutlu, O.; Ghose, S.; Gómez-Luna, J.; Ausavarungnirun, R. A modern primer on processing in memory. In *Emerging Computing: From Devices to Systems: Looking Beyond Moore and Von Neumann*; Springer: Berlin/Heidelberg, Germany, 2022; pp. 171–243.
12. Bavikadi, S.; Sutradhar, P.R.; Indovina, M.A.; Ganguly, A.; Dinakarrao, S.M.P. Reapprox-pim: Reconfigurable approximate lookup-table (lut)-based processing-in-memory (pim) machine learning accelerator. *IEEE Trans. Comput.-Aided Des. Integr. Circuits Syst.* **2024**, *43*, 2288–2300. [CrossRef]
13. Song, L.; Zhuo, Y.; Qian, X.; Li, H.; Chen, Y. GraphR: Accelerating graph processing using ReRAM. In *Proceedings of the 2018 IEEE International Symposium on High Performance Computer Architecture (HPCA)*; IEEE: New York, NY, USA, 2018; pp. 531–543.
14. Challapalle, N.; Rampalli, S.; Song, L.; Chandramoorthy, N.; Swaminathan, K.; Sampson, J.; Chen, Y.; Narayanan, V. GaaS-X: Graph analytics accelerator supporting sparse data representation using crossbar architectures. In *Proceedings of the 2020 ACM/IEEE 47th Annual International Symposium on Computer Architecture (ISCA)*; IEEE: New York, NY, USA, 2020; pp. 433–445.
15. Zhou, M.; Imani, M.; Gupta, S.; Kim, Y.; Rosing, T. Gram: Graph processing in a reram-based computational memory. In Proceedings of the IEEE Asia and South Pacific Design Automation Conference, Tokyo, Japan, 21–24 January 2019.
16. Jin, H.; Chen, D.; Zheng, L.; Huang, Y.; Yao, P.; Zhao, J.; Liao, X.; Jiang, W. Accelerating graph convolutional networks through a PIM-accelerated approach. *IEEE Trans. Comput.* **2023**, *72*, 2628–2640. [CrossRef]
17. Ahmed, A.; Lin, F.; Li, J.; Skadron, K. TGN-PNM: A Near-Memory Architecture for Temporal GNN Inference on 3D-Stacked Memory. In Proceedings of the International Symposium on Memory Systems, Washington, DC, USA, 7–9 October 2025; pp. 177–189.
18. Yang, S.; He, S.; Wang, W.; Yin, Y.; Wu, T.; Chen, W.; Zhang, X.; Sun, X.H.; Feng, D. GOPIM: GCN-oriented pipeline optimization for PIM accelerators. In *Proceedings of the 2025 IEEE International Symposium on High Performance Computer Architecture (HPCA)*; IEEE: New York, NY, USA, 2025; pp. 657–670.
19. Deng, Q.; Zhang, Y.; Zhang, M.; Yang, J. Lacc: Exploiting lookup table-based fast and accurate vector multiplication in dram-based cnn accelerator. In Proceedings of the 56th Annual Design Automation Conference 2019, Las Vegas, NV, USA, 2–6 June 2019; pp. 1–6.
20. Ferreira, J.D.; Falcao, G.; Gómez-Luna, J.; Alser, M.; Orosa, L.; Sadrosadati, M.; Kim, J.S.; Oliveira, G.F.; Shahroodi, T.; Nori, A.; et al. pLUTo: Enabling massively parallel computation in DRAM via lookup tables. In *Proceedings of the 2022 55th IEEE/ACM International Symposium on Microarchitecture (MICRO)*; IEEE: New York, NY, USA, 2022; pp. 900–919.
21. Gu, P.; Xie, X.; Li, S.; Niu, D.; Zheng, H.; Malladi, K.T.; Xie, Y. DLUX: A LUT-based near-bank accelerator for data center deep learning training workloads. *IEEE Trans. Comput.-Aided Des. Integr. Circuits Syst.* **2020**, *40*, 1586–1599. [CrossRef]
22. Ramanathan, A.K.; Kalsi, G.S.; Srinivasa, S.; Chandran, T.M.; Pillai, K.R.; Omer, O.J.; Narayanan, V.; Subramoney, S. Look-up table based energy efficient processing in cache support for neural network acceleration. In *Proceedings of the 2020 53rd Annual IEEE/ACM International Symposium on Microarchitecture (MICRO)*; IEEE: New York, NY, USA, 2020; pp. 88–101.
23. Sutradhar, P.R.; Bavikadi, S.; Connolly, M.; Prajapati, S.; Indovina, M.A.; Dinakarrao, S.M.P.; Ganguly, A. Look-up-table based processing-in-memory architecture with programmable precision-scaling for deep learning applications. *IEEE Trans. Parallel Distrib. Syst.* **2021**, *33*, 263–275. [CrossRef]
24. Kipf, T. Semi-supervised classification with graph convolutional networks. *arXiv* **2016**, arXiv:1609.02907.
25. Xu, K.; Hu, W.; Leskovec, J.; Jegelka, S. How powerful are graph neural networks? *arXiv* **2018**, arXiv:1810.00826.

26. Esfahani, M.K.; Kilpatrick, P.; Vandierendonck, H. Locality analysis of graph reordering algorithms. In *Proceedings of the 2021 IEEE International Symposium on Workload Characterization (IISWC)*; IEEE: New York, NY, USA, 2021; pp. 101–112.
27. Wheatman, B.; Xu, H. Packed compressed sparse row: A dynamic graph representation. In *Proceedings of the 2018 IEEE High Performance Extreme Computing Conference (HPEC)*; IEEE: New York, NY, USA, 2018; pp. 1–7.
28. Alawneh, T.A.; Jarajreh, M.M.; Alkasassbeh, J.S.; Sharadqh, A.A. High-performance and power-saving mechanism for page activations based on full independent DRAM sub-arrays in multi-core systems. *IEEE Access* **2023**, *11*, 79801–79822. [CrossRef]
29. Tian, T.; Wang, X.; Zhao, L.; Wu, W.; Zhang, X.; Lu, F.; Wang, T.; Jin, X. G-NMP: Accelerating graph neural networks with DIMM-based near-memory processing. *J. Syst. Archit.* **2022**, *129*, 102602. [CrossRef]
30. Giannoula, C.; Yang, P.; Fernandez, I.; Yang, J.; Durvasula, S.; Li, Y.X.; Sadrosadati, M.; Luna, J.G.; Mutlu, O.; Pekhimenko, G. PyGim: An Efficient Graph Neural Network Library for Real Processing-In-Memory Architectures. *Proc. ACM Meas. Anal. Comput. Syst.* **2024**, *8*, 1–36. [CrossRef]
31. Singh, G.; Kuppannagari, S.R.; Vrudhula, S. PARAG: PIM Architecture for Real-Time Acceleration of GCNs. In *Proceedings of the 2023 IEEE 30th International Conference on High Performance Computing, Data, and Analytics (HiPC)*; IEEE: New York, NY, USA, 2023; pp. 11–20.
32. Chen, D.; Li, H.; Lan, H.; Li, Z.; Yao, P.; Mitra, T. HighP: In-Memory Acceleration of SpGEMM with High Bank-Level Parallelism. *IEEE Trans. Comput.* **2025**, *75*, 139–151. [CrossRef]
33. Zeng, H.; Zhou, H.; Srivastava, A.; Kannan, R.; Prasanna, V. Graphsaint: Graph sampling based inductive learning method. *arXiv* **2019**, arXiv:1907.04931.
34. Karypis, G.; Kumar, V. *A Software Package for Partitioning Unstructured Graphs, Partitioning Meshes, and Computing Fill-Reducing Orderings of Sparse Matrices*; University of Minnesota, Department of Computer Science and Engineering, Army High Performance Computing Research Center: Minneapolis, MN, USA, 1998; Volume 38, pp. 1–7.
35. Yuan, H.; Liu, Y.; Zhang, Y.; Ai, X.; Wang, Q.; Chen, C.; Gu, Y.; Yu, G. Comprehensive evaluation of gnn training systems: A data management perspective. *arXiv* **2023**, arXiv:2311.13279.
36. Seshadri, V.; Kim, Y.; Fallin, C.; Lee, D.; Ausavarungnirun, R.; Pekhimenko, G.; Luo, Y.; Mutlu, O.; Gibbons, P.B.; Kozuch, M.A.; et al. RowClone: Fast and energy-efficient in-DRAM bulk data copy and initialization. In Proceedings of the 46th Annual IEEE/ACM International Symposium on Microarchitecture, Davis, CA, USA, 7–11 December 2013; pp. 185–197.
37. Chang, K.K.; Nair, P.J.; Lee, D.; Ghose, S.; Qureshi, M.K.; Mutlu, O. Low-cost inter-linked subarrays (LISA): Enabling fast inter-subarray data movement in DRAM. In *Proceedings of the 2016 IEEE International Symposium on High Performance Computer Architecture (HPCA)*; IEEE: New York, NY, USA, 2016; pp. 568–580.
38. Kim, Y.; Seshadri, V.; Lee, D.; Liu, J.; Mutlu, O. A case for exploiting subarray-level parallelism (SALP) in DRAM. *ACM SIGARCH Comput. Archit. News* **2012**, *40*, 368–379. [CrossRef]
39. Keeth, B.; Baker, R.J.; Johnson, B.; Lin, F. *DRAM Circuit Design: Fundamental and High-Speed Topics*; John Wiley & Sons: Hoboken, NJ, USA, 2007.
40. Fey, M.; Lenssen, J.E. Fast graph representation learning with PyTorch Geometric. *arXiv* **2019**, arXiv:1903.02428.

Disclaimer/Publisher's Note: The statements, opinions and data contained in all publications are solely those of the individual author(s) and contributor(s) and not of MDPI and/or the editor(s). MDPI and/or the editor(s) disclaim responsibility for any injury to people or property resulting from any ideas, methods, instructions or products referred to in the content.