

MOSAIC: An Accelerator Design for Heterophilic Graph Learning

Fangzhou Ye

University of Central Florida
Orlando, FL, USA
fangzhou.ye@ucf.edu

Wei Zhang

University of Central Florida
Orlando, FL, USA
wzhang.cs@ucf.edu

Ahmed Louri

George Washington University
Washington, DC, USA
louri@email.gwu.edu

Hao Zheng

University of Central Florida
Orlando, FL, USA
hao.zheng@ucf.edu

Abstract—Heterophilic graphs are common in many real-world applications such as protein-protein interaction networks, where connected nodes often have dissimilar features. Traditional Graph Neural Networks (GNNs) operate exclusively on immediate neighbors and extend the receptive field through layer expansion, failing to capture crucial distant relationships and structural information due to the over-smoothing problem. Recent theoretical developments of higher-order GNNs are shifting towards K-hop feature aggregation, where each vertex can simultaneously aggregate information from distant vertices across several hops. Despite their theoretical promise, higher-order GNNs suffer from significant memory bottlenecks and can be up to 100× slower than traditional GNNs. This inefficiency arises primarily from (1) the requirement to maintain separate adjacency matrices for different hop distances (i.e., powers of the adjacency matrix), and (2) redundant computation and memory accesses caused by overlapping paths during multi-hop aggregation.

In this paper, we propose a clique-centric accelerator design, called MOSAIC, that aims to decouple the computation and memory overheads of K-hop aggregation from the expanded receptive field. Our key observation is that clique graphs are the primitive components that capture the overlapping paths of distant feature aggregations. Based on the clique structure, we develop a framework to accelerate higher-order GNNs, achieving performance and latency comparable to traditional GNN models. First, we propose a clique-centric dataflow to eliminate the redundant computation and memory access caused by path overlapping due to simultaneous node aggregations with distinct distances. Second, we propose a K-hop degree partitioning algorithm that can incorporate the degree information of the extended receptive field with a balanced workload. Third, we propose a new data representation that can encode the graph connectivity of nodes with different distances, eliminating the need for separate adjacency matrices. Lastly, we propose a novel accelerator architecture to enable clique reuse as opposed to recomputing feature aggregation during concurrent aggregations. Our simulation results show an average speedup of 7.67×, 6.73×, and 4.88× over I-GCN [1], Betty [2], and ReGNN [3] for higher-order GNNs, respectively. Meanwhile, MOSAIC achieves energy reduction by 51.78%, 52.53%, and 56.84% on average compared with the baselines.

Index Terms—Higher-order GNN, heterophilic graph learning, multi-hop aggregation, accelerator.

I. INTRODUCTION

Graph Neural Networks (GNNs) [4]–[9] have been widely used to learn the relationships between entities in many application domains, such as social networks [10], [11], biological

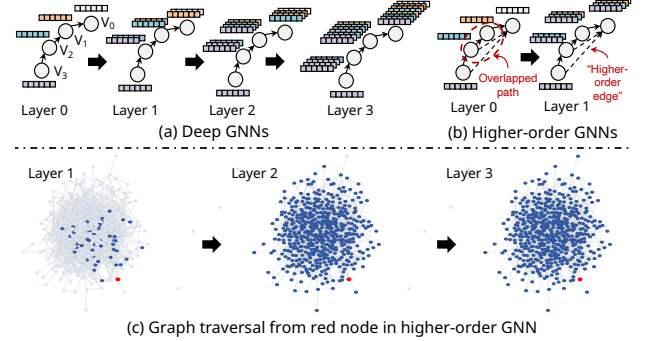


Fig. 1. Comparison of (a) Deep GNNs and (b) Higher-order GNNs with 3-hop. (c) The graph traversal path (e.g., feature aggregation) from the red node, where the darker color indicates traversed nodes.

networks [12]–[14], communication networks [15], [16], recommendation systems [17], [18], among many others. Existing GNNs typically perform message passing in a layer-wise fashion, aggregating information from immediate neighbors at each layer as shown in Figure 1(a). However, layer-wise feature aggregation is inefficient in capturing structural information in heterophilic graphs [19]–[21], where connected nodes often exhibit dissimilarity. The heterophilic nature of graphs is frequently observed in biological networks, particularly in the domain of molecular biology [22]–[25]. Consequently, recent theoretical development [26], [27] has been shifting towards higher-order GNNs to enable distant feature aggregation, such as MixHop [26], where vertex features from multiple hops are simultaneously concatenated to distinguish vertex similarity, as shown in Figure 1(b). The K-hop aggregation significantly enlarges the receptive field, covering a substantial number of nodes as shown in Figure 1(c). This leads to up to 100× slowdown in execution time compared to traditional multi-layer GNN models [28]–[30].

The significant performance overhead of higher-order GNNs arises from aggregating features from distant nodes simultaneously, which is called K-hop aggregation. This requires using powers of the adjacency matrices to capture higher-order graph connectivity, resulting in considerable path overlap during aggregation. Traditional deep GNNs expand the receptive field through layer expansion, where embedding features are propagated from hop to hop. While layer-wise expansion is efficient for recommendation and security systems [28] (e.g., Pinterest PinSage [31], JPMorgan Chase JEL [32]), the

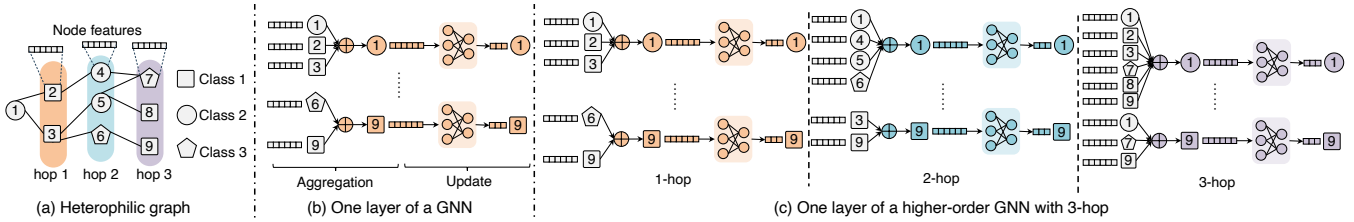


Fig. 2. An example of a heterophilic graph, traditional single-layer GNNs, and higher-order GNNs.

over-smoothing issue can make distant graph structure indistinguishable. While prior work [33]–[40] has attempted to optimize the performance of deep GNNs with layer expansion, the computational challenges of higher-order GNNs remain relatively under-explored. For example, traditional GNNs only need one-hop connectivity information to aggregate features from immediate neighbors (e.g., adjacency matrix). PyG [33] and its variants [34], [41] implement higher-order GNNs by leveraging powers of the adjacency matrices to capture higher-degree graph connectivity, leading to additional computation and memory overheads.

Moreover, prior works [1]–[3], [42]–[44] exploit various graph theories to eliminate redundant feature aggregations or to regularize the adjacency matrix, optimizing GNN performance and memory efficiency. However, most of them assume that nodes aggregate features only from their immediate neighbors, while higher-order GNNs expand the receptive field by incorporating information from distant nodes. For example, GNNAdvisor [43] requires pre-processing to reorder the adjacency matrix to regularize sparse-dense matrix multiplication (SpMM) [45], [46]. Similarly, I-GCN [1] utilizes a breadth-first search strategy to uncover dense graph connectivity, aiming to enhance the computational regularity of the adjacency matrix. ReGNN [3], on the other hand, introduces a dynamic message-passing mechanism that eliminates redundancy in neighborhoods to reduce unnecessary computation during feature aggregation from directly connected nodes. Redundant computations and memory accesses caused by substantial path overlap in K-hop aggregation have yet to be thoroughly explored and addressed.

In this paper, we aim to identify redundant computation and memory access in higher-order graph neural networks. We observe that the clique structure is the primitive graph component that can encode the possible path-overlapping redundancy in higher-order graph neural networks. Reusing the clique structure can fundamentally eliminate redundant computation and memory access in K-hop aggregation. Upon this key insight, we propose a clique-centric accelerator design.

This paper makes the following contributions:

- **Clique-Centric Computation for K-hop Aggregation:** We propose a clique-centric computation to address redundant computation and memory access in higher-order GNNs. The proposed algorithm operates on the clique structure as opposed to traditional vertex-centric GNNs.
- **Dataflow Optimization for Higher-order GNNs:** We propose a clique-centric dataflow optimization to parallelize higher-order GNNs. Specifically, we propose a K-

hop graph traversal algorithm that can evenly partition the raw graph data based on K-hop degree and clique count, which can balance the computation of K-hop aggregation. In addition, we propose a new dataflow optimization that can minimize off-chip memory access for the proposed clique-centric computation.

- **Novel Accelerator Architecture with KV Cache for Clique Reuse:** We propose a novel accelerator architecture for higher-order GNNs to support the reuse of graph cliques, eliminating redundant computations for both neighborhood and path-overlapping aggregations. We further propose a K-order list that eliminates the need for storing the powers of graph adjacency matrices while encoding the path-overlapping redundancy information. Lastly, the proposed PE architecture enables in-unit aggregation and update phases that can significantly reduce the memory access for intermediate data.

Our simulation results show an average speedup of $7.67\times$, $6.73\times$, and $4.88\times$ over I-GCN [1], Betty [2], and ReGNN [3] in terms of support for higher-order GNNs, respectively. Meanwhile, MOSAIC achieves energy reduction by 51.78%, 52.53%, and 56.84% compared with baselines.

II. BACKGROUND

A. Heterophilic Graph

Graphs are widely used to express the relationship between entities, which can be categorized into two types: heterophilic graphs (where similar nodes are not directly connected but may be linked through longer paths) and homophilic graphs (where connected nodes tend to have similar attributes or belong to the same class). For example, in homophilic graphs such as citation networks and social networks, authors who cite similar papers or friends with shared interests tend to form tightly knit communities. Conversely, many real-world graphs display heterophily [19], [21], as seen in biological networks, where proteins with similar functions may not interact directly but can affect each other through multi-step biochemical pathways. For example, as shown in Figure 2(a), node 4, which shares the same class as node 1, is a 2-hop neighbor rather than an immediate neighbor. The heterophilic graphs introduce new challenges for graph neural networks.

B. Graph Neural Networks

Graph Neural Networks (GNNs) [47]–[49] have emerged as a promising model for extracting knowledge from graph data. GNN models are composed of multiple layers, each of which consists of the aggregation and update phases.

TABLE I
SUMMARY OF HIGHER-ORDER GNN APPLICATIONS, MODELS, HOP
SETTINGS, AND TYPICAL GRAPH DENSITY.

Application Domain	Higher-Order GNN Model	# Hops (K)	Graph Density
Node Classification	MixHop [26]; H2GCN [52]; FSGNN [53]	$K = 2-4$	$10^{-7}-10^{-2}$
Fraud/Anomaly Detection	SEC-GFD [54]; H2GCN [52]	$K = 2-6$	$10^{-5}-10^{-2}$
Drug-Drug Interaction (DDI)	GNN-DDI [55]	$K = 5$	$10^{-2}-10^{-1}$
Traffic Flow Forecasting	HopFormer [56]; SST-GNN [57]	$K = 2-3$	$10^{-3}-10^{-2}$
Knowledge Graph Reasoning	GCN-JK / GAT-JK [58]; MAGNA [59]	$K = 2-6$	$10^{-3}-10^{-2}$
Molecular Property Prediction	MPNN [60]	$K = 2-5$	$10^{-2}-10^{-1}$

GNNs commonly employ a message-passing approach [50] to exchange vertex and edge features, followed by neural network operations. For example, as shown in Figure 2(b), given a graph $G = (V, E)$ and in layer l , the definition of the message-passing mechanism of GNNs in the aggregation phase is formulated as follows:

$$\mathcal{M}_{\mathcal{N}(v)}^{(l)} = \text{AGGREGATE}(h_u^{(l)}, \forall u \in \mathcal{N}(v)) \quad (1)$$

where the *Aggregate* function aggregates multiple feature vectors from v 's neighbors ($\mathcal{N}(v)$) hidden embeddings $h_u^{(l)}$ correspond to each node $u \in \mathcal{N}(v)$ to “message” $\mathcal{M}_{\mathcal{N}(v)}^{(l)}$. The update phase in each GNN layer can be formulated as:

$$h_v^{l+1} = \text{UPDATE}(h_v^l, \mathcal{M}_{\mathcal{N}(v)}^{(l)}) \quad (2)$$

where $\mathcal{M}_{\mathcal{N}(v)}^{(l)}$ is based on its aggregated neighborhood information. The *Update* function combines aggregated features with the prior embedding $h_v^{(l)}$ of node v to produce the refreshed embedding $h_v^{(l+1)}$. Despite rapid advancement in GNNs, current models are mostly optimized for homophilic graphs such as social networks, where their receptive field is extended through layer expansion. However, multi-layer GNNs are unable to preserve the global structural information of heterophilic graphs due to over-smoothing [51].

C. Higher-order Graph Neural Networks

Higher-order GNNs [7], [26], [27], [58], [59], [61], [62], such as MixHop [26] and Multi-hop Attention Graph Neural Network (MAGNA) [59], extend the aggregation range of traditional multi-layer GNNs to capture long-distance neighbors (i.e., K -hop aggregation), making GNNs suitable for learning on heterophilic graphs. By incorporating information from distant neighbors, higher-order GNNs can better capture the global structure of the graph and enhance the learning of meaningful node representations [52], [63], [64].

Higher-order GNNs aggregate features from distant nodes, as illustrated in Figure 2(c). While higher-order GNNs share similarities with traditional GNNs in the update phase, they perform separate aggregation and update operations for each hop [65]. Given the same graph $G(V, E)$, the definition of the aggregation phase is as follows:

$$\mathcal{M}_{\mathcal{N}_k(v)}^{(l)} = \text{AGGREGATE}(\{h_u^l, \forall u \in \mathcal{N}_k(v)\}) \quad (3)$$

here, $\mathcal{N}_k(v)$ are K -hop neighbors of vertex v and $\mathcal{M}_{\mathcal{N}_k(v)}^{(l)}$ is the aggregated vertex features in hop k . h_u^l denotes the distant vertex in hop k . However, higher-order GNNs need the powers of adjacency matrices to represent the graph connectivity

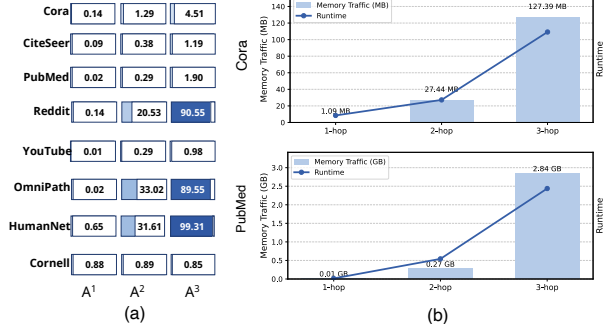


Fig. 3. (a) Density ratio and patterns of higher-order adjacency matrices. Darker colors indicate higher values. (b) Memory access comparison.

of vertices at different distances, leading to memory and computation overhead, as shown in Figure 3(b). Furthermore, as shown in Figure 3(a), the severe workload imbalance across different hops hinders parallel execution, introducing poor performance and computation resource waste.

After the aggregation, the definition of the final output with vertex update is represented as:

$$h_v^{(l+1)} = \text{MERGE}_{k=1}^K \left(\text{UPDATE}(h_v^l, \mathcal{M}_{\mathcal{N}_k(v)}^{(l)}) \right) \quad (4)$$

The outputs from each K -hop neighborhood aggregation and update are merged by *Merge* functions. For example, MixHop GNN uses the *concat* function to concatenate outputs, while Column Networks [66] interpolate the neighborhood representation and the node’s previous representation.

Higher-order GNN Applications. Satisfying higher-order GNN requirements typically necessitates extending the receptive field to $K = 2-4$ hops on average across application domains. Table I summarizes the applications and required receptive field of higher-order GNN models. For example, extending the receptive field in fraud detection tasks [54] reflects deep fraudulent camouflage via multi-hop paths. Multi-hop MPNN [60] captures bond-ring structures (e.g., detecting 6-membered rings requires at least $K = 3$) in molecular property and ADMET prediction tasks. Knowledge graph reasoning [59] traverses deeper neighbor relationships to construct more coherent and logically grounded results. Furthermore, graphs used in higher-order GNN workloads are typically sparse. Among commonly used benchmarks, DDI graphs can reach densities of approximately 1.4×10^{-1} , representing one of the densest higher-order GNN workloads reported in the literature [67].

D. Challenges of Higher-order GNNs

Even though higher-order GNNs [52], [63], [64] are capable of learning from heterophilic graphs by explicitly aggregating features beyond immediate neighbors, they introduce several distinct computational challenges.

1) Feature Aggregation Redundancy: Redundant aggregation is a common issue in traditional GNNs due to graph connectivity, as shown in Figure 4. Prior works [1], [3] mainly focus on eliminating redundant computations when aggregating immediate neighbors. Such redundant computations result from the mutually shared neighbors between vertices, which

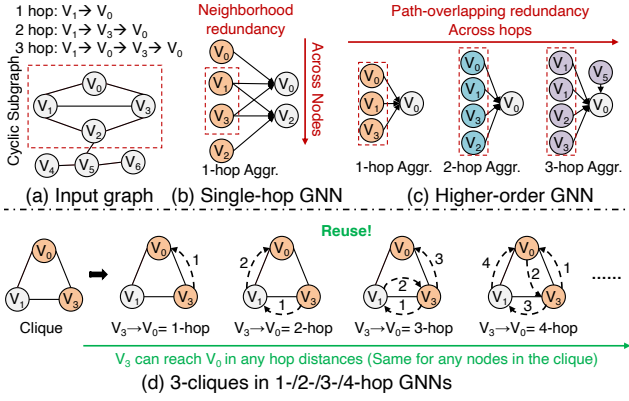


Fig. 4. Different types of redundancy and reuse opportunities in single-hop and higher-order GNNs.

we call neighborhood redundancy. However, the traversal paths to distant nodes significantly overlap in K-hop aggregation, which contributes up to 99% redundant computations in the aggregation phase.

Neighborhood redundancy occurs when multiple source nodes share common neighbors. For instance, as shown in Figure 4(b), a pair of nodes, V_1 and V_3 , are directly connected to both V_0 and V_2 . In traditional GNNs, nodes V_0 and V_2 receive identical aggregation results due to this shared neighborhood. Recent research [1]–[3] has addressed neighborhood redundancy by identifying commonly shared neighbor pairs or densely connected graph components. For example, ReGNN [3] introduces a redundancy-free neighborhood message-passing scheme by identifying and aggregating mutually shared node pairs prior to each hop aggregation phase. Similarly, I-GCN [1] reorganizes the adjacency matrix by clustering strongly connected nodes into localized “islands”. However, both methods are limited to aggregation between immediate neighbors.

Path-overlapping redundancy is unique to higher-order GNNs where aggregation paths to distant nodes often exhibit significant overlap. For example, as shown in Figure 4(a), when node V_0 operates 3-hop feature aggregation (i.e., simultaneously fetching features from nodes located at 1-, 2-, and 3-hop distances), node V_1 is located in all the aggregation paths, as illustrated on top of the cyclic subgraph. As depicted in Figure 4(c), V_0 redundantly aggregates the same set of vertices (e.g., V_0, V_1, V_3) in 3-hop aggregations.

In contrast to neighborhood redundancy, path overlap during K-hop aggregation arises from cyclic structures. When traversing a cyclic graph from a common source node, paths may repeatedly visit the same nodes. As such, during K-hop aggregation - where each vertex gathers features from nodes that are K-hop away - these traversal paths often overlap significantly. Figure 4(d) illustrates that any node in the clique could reach all nodes (including itself) within any hop distance, indicating that the aggregation results of cliques could be reused persistently. This motivates us to identify cliques as primitive components to avoid redundant aggregation in higher-order GNNs.

III. PROPOSED ALGORITHMIC OPTIMIZATION

Despite significant efforts [1], [43], [44], higher-order aggregation introduces new challenges in workload partitioning, dataflow, and redundancy elimination. For example, vertex or edge counts are typically used to balance the aggregation and update phases of traditional GNNs. However, simply counting edge or vertex numbers is unable to capture the workload of aggregating features from distant nodes due to path-overlapping redundancy and varying higher-order graph connectivity. Furthermore, traditional GNNs rely on reordering the adjacency matrix or mutating loop orders to reduce memory access. However, higher-order GNNs require the powers of adjacency matrices, where the graph connectivity and density vary significantly when the receptive field extends. Consequently, prior studies focused primarily on 1-hop aggregation offer limited insights for optimizing memory access in K-hop aggregation.

A. Clique-centric Algorithm to Eliminate K-hop Redundancy

Currently, higher-order GNNs are implemented in PyG [33] and DGL [41] using both sequential and parallel models, where adjacency matrix powers are computed in proportion to the receptive field distance. The separate representations of higher-order graph connectivity overlook the path-overlapping information. The path-overlapping redundancy of K-hop aggregation is mainly caused by the cyclic graph connectivity, in which the chained vertices form redundant inclusion of each other in long-distance graph traversal. This is evidenced by the increased density in the higher powers of the adjacency matrix, as shown in Figure 3. A clique in a graph is a subset of its vertices such that every two distinct vertices are connected by an edge, forming a complete graph. Each vertex needs to traverse all connected vertices, and therefore, all vertices share the same K-hop traversal paths. Such a complete graph structure not only causes path-overlapping but also neighborhood redundancy. Based on our preliminary results illustrated in Figure 5(a), path-overlapping and neighborhood redundancy contribute to 95.19% of the total K-hop aggregation across different datasets on average.

We aim to identify the clique structure to eliminate redundant computation and memory access for K-hop aggregation. However, a clique can consist of any number of vertices, and determining the maximum number of cliques [68] in a graph is known to be an NP-hard problem, which can increase the search time exponentially. Therefore, we repurpose the well-known triangle detection algorithms [69], where finding a 3-clique only requires polynomial time. In addition, a 3-clique is the primitive component of any clique structure. For example, as shown in Figure 5(b), 3-cliques make up 74.11% of the total number of 3-, 4-, and 5-cliques combined. Theoretically, each 3-clique structure only needs to be computed once, and its intermediate results (IR) can be used for any connected vertices. For instance, as illustrated in Figure 5(c), (V_0, V_1, V_2) forms a 3-clique, and the aggregated results can be reused to eliminate both neighborhood and path-overlapping redundancy.

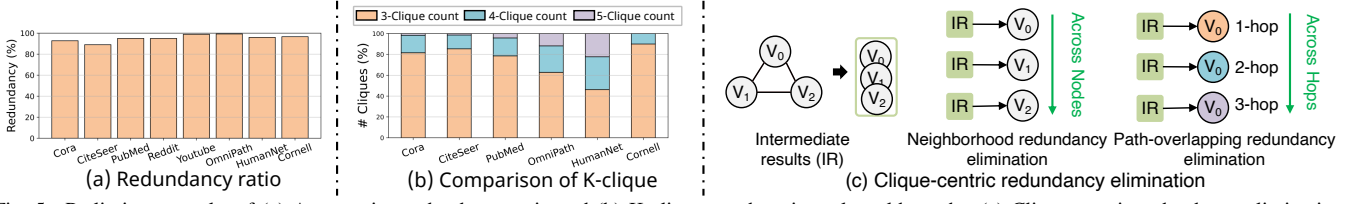


Fig. 5. Preliminary results of (a) Aggregation redundancy ratio and (b) K-clique numbers in real-world graphs. (c) Clique-centric redundancy elimination.

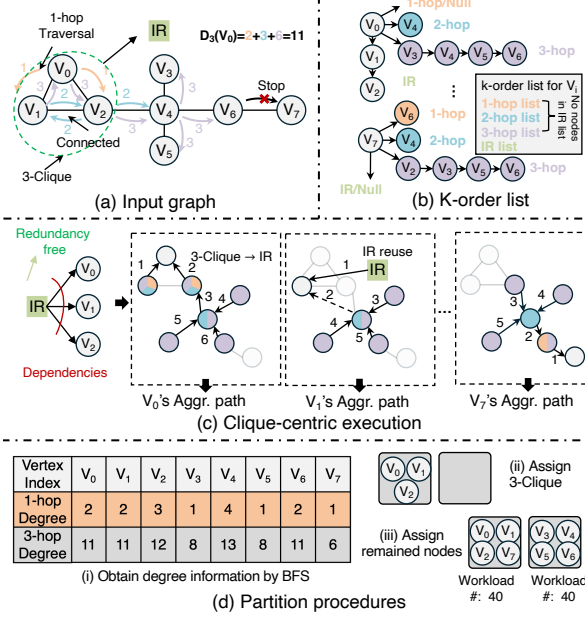


Fig. 6. An example of Clique-centric execution flow and partition algorithm.

Clique-centric Computation for Higher-order GNN. Unlike traditional GNNs, directly pursuing edge- or node-centric parallelism could lead to redundant computations in higher-order GNNs due to path overlapping. However, the node dependency within each clique structure in turn affects the parallelism. For example, as shown in Figure 6, one node generates the intermediate representation (IR), while the others in the clique must wait to reuse the result.

To address this issue, we propose a clique-centric dataflow for higher-order GNNs. The key idea is prioritizing the execution of nodes belonging to cliques, while staggering the processing of nodes within the same clique to avoid redundancy. This ensures that intermediate aggregation results are reused efficiently, eliminating redundant computations. In other words, each clique is the primitive structure for K-hop aggregation, which only needs to be calculated once. For example, as shown in Figure 6(c), a 3-clique, including nodes V_0, V_1 , and V_2 , is scheduled for K-hop aggregation following the graph traversal sequence (which we will discuss in the next section). We will start processing the 1-hop aggregation for V_0 to obtain the aggregated results of V_0, V_1 , and V_2 . Afterward, the K-hop aggregations of all nodes V_0, V_1 , and V_2 can be executed in parallel.

K-order List. The next problem is the data representation of the graphs and the compression format. Conventional GNNs use an adjacency matrix to represent one-hop graph connectivity, which is stored in a sparse format such as a compressed

Algorithm 1: 3-clique Listing with K-hop BFS

Input : Graph: $G = (V, E)$; Number of Execution Hops: k
Output: 3-cliques: C ; K-hop Degree: D

```

1  $C \leftarrow \emptyset, D \leftarrow \text{zeros}(|V|)$ ;
2 for  $v \in V$  do
3    $\text{visited} \leftarrow \text{set}(), \text{queue} \leftarrow \text{Queue}()$ ;
4    $\text{queue.push}((v, 0))$ ;
5    $\text{one\_hop} \leftarrow []$  // Collect 1-hop neighbors of  $v$ 
6   while  $\text{queue}$  is not empty do
7      $(\text{node}, \text{depth}) \leftarrow \text{queue.pop}()$ ;
8     if  $\text{node} \in \text{visited}$  or  $\text{depth} > K$  then
9       continue
10     $\text{visited.add}(\text{node})$ ;
11     $D[v] += 1$  // Update degree for node  $v$ 
12    if  $\text{depth} == 1$  then
13      // Check 3-clique
14       $\text{one\_hop.add}(\text{node})$ ;
15      for  $u \in \text{one\_hop}$  do
16        if  $\text{node} \in N(u)$  then
17           $C \leftarrow C \cup \{(v, u, \text{node})\}$ ;
18           $\text{one\_hop.remove}(\text{node}, u)$ ;
19      for  $\text{neighbor} \in N(\text{node})$  and  $\text{neighbor} \notin \text{visited}$  do
20         $\text{queue.push}((\text{neighbor}, \text{depth} + 1))$ ;
21 return  $C, D$ 

```

sparse row (CSR) [70]. However, the traditional adjacency matrix is unable to encode higher-order graph connectivity and redundancy information. For example, computing the powers of the adjacency matrix incurs prohibitive storage costs. To solve this problem, we propose a K-order list structure for higher-order GNNs, which incorporates both higher-order graph connectivity (i.e., K-hop lists without clique nodes) and clique structures (i.e., an IR list). The proposed K-order list for each vertex includes its K-hop neighboring nodes and detected cliques. For instance, as illustrated in Figure 6(b), V_0 has 1-hop, 2-hop, and 3-hop lists to store non-clique nodes (the nodes not in the IR list), where the 1-hop list is Null, as well as an IR list to store clique nodes.

K-hop Breadth-first Traversal for 3-clique Search and K-order List. As mentioned, both neighborhood and path-overlapping redundancy can be efficiently eliminated by the 3-clique structure and the K-order list. However, it is computationally prohibitive to acquire cliques and generate the K-order list by computing multiple powers of the adjacency matrix. As such, we propose a K-hop breadth-first traversal (BFS) algorithm, as illustrated in Algorithm 1. The purpose of K-hop BFS is twofold: identifying 3-cliques (Figure 6(a)) and retrieving the K-order list (Figure 6(b)). Specifically, node V_0 first traverses its neighbors within k hops, such as 1-hop neighbors (V_1 and V_2), to construct a K-order list. Simultane-

ously, the connectivity of V_1 and V_2 will be checked, recording the 3-clique graphlet (V_0, V_1, V_2) (Algorithm 1 Lines 11-17). This process continues for V_0 until it reaches the nodes at 3 hops away, which means V_0 will not access its 4-hop neighbor V_7 . To optimize triangle detection, we employ a greedy strategy during graph traversal to heuristically maximize the total number of 3-cliques. Each node may belong to multiple cliques, and therefore, traversal paths determine the number of unique cliques that can be found. We reorder the graph using 1-hop degree information, prioritizing traversal from nodes with smaller degrees. This is due to the fact that higher graph connectivity often results in suboptimal 3-clique searches because of the large number of possible paths.

B. Parallelism Algorithm for Clique-centric Computation

Even though workload partitioning [71], [72] has been well studied in GNN acceleration, existing solutions are often optimized to evenly partition the workload (e.g., vertices or edges) related to the input graph. For example, BGL [71] evenly partitions vertices into contiguous memory blocks (i.e., chunks) to balance the vertices' workload while simplifying memory access patterns. On the other hand, MGG [72] aims to partition edges evenly across graph partitions. However, higher-order GNNs involve distant neighbors, meaning that simply balancing vertices or edges does not necessarily ensure balance during the aggregation or update phases. The workload for higher-order GNNs is related to the K-hop degree, which is defined as:

$$D_k(v) = \sum_{k=1}^K |\mathcal{N}_k(v)| \quad (5)$$

Additionally, prior algorithms [1], [3] are tailored for one-hop aggregation and preprocess the input graphs to capture one-hop connectivity. As a result, adapting ReGNN and I-GCN for K-hop aggregation necessitates iterative preprocessing to capture higher-order graph connectivity.

We propose a clique-centric partition algorithm that is utilized in raw graphs to explore maximum redundancy elimination while maintaining workload balance with K-hop degree. The proposed partitioning algorithm is based on the previously mentioned BFS algorithm, as illustrated in Figure 6. Specifically, we model this as a variant of the bin-packing problem, where each "bin" represents a graph partition containing a set of vertices. Unlike traditional bin packing, which minimizes the number of bins, our approach evenly distributes the workload across a fixed number of partitions. The total number of bins corresponds to the number of workload partitions, and each bin's capacity is determined by dividing the total number of vertices by the number of partitions. For example, when V_0 traverses its 1-hop neighbors V_1 and V_2 , the K-hop degree of V_0 can be obtained. Subsequently, the clique-centric partition algorithm creates workload-balanced task groups in 2 steps. Firstly, as illustrated in Algorithm 2 (Lines 13-15), triangles will be assigned to different bins, where each triangle has a similar K-hop degree. For example, as shown in Figure 6(d), the 3-clique (V_0, V_1, V_2) is assigned to the same bin. Similarly,

Algorithm 2: Clique-centric Parallelism Optimization

Input : Graph: $G = (V, E)$; Number of Partitions: N ;
Number of Execution Hops: k ; Degrees of K-hop:
 D ; 3-cliques: C

Output: Workload Partitions: T

```

1  $V_{avg} \leftarrow |V|/N$ ,  $R_v \leftarrow |V| \bmod N$ ,  $T \leftarrow \emptyset$ ,  

    $Cap \leftarrow \sum_{v \in V} D_k(v)/N$ ;
2 Function FindBestFit( $D_{nodes}$ ,  $T$ ,  $V_{avg}$ ,  $R_v$ ,  $Cap$ )
3    $best\_place \leftarrow \text{None}$ ,  $min\_wl\_dif \leftarrow \infty$ ;
4   for  $T_n \in T$  do
5     if  $(|T_n| < V_{avg})$  or  $(|T_n| = V_{avg} \text{ and } R_v > 0)$  then
6        $wl\_dif \leftarrow Cap_n - |T_n \cdot D| - D_{nodes}$ ;
7       if  $wl\_dif \geq 0$  and  $wl\_dif < min\_wl\_dif$  then
8          $best\_place \leftarrow n$ ,  $min\_wl\_dif \leftarrow wl\_dif$ ;
9   if  $best\_place = \text{None}$  then
10     $best\_place \leftarrow \arg \min_n (T_n \cdot D)$ ;
11   return  $best\_place$ ;
12 // Step 1: Assign 3-cliques to Partitions
13 for  $(u, v, w) \in C$  do
14    $best\_place \leftarrow \text{FindBestFit}(D_{u,v,w}, T, V_{avg}, R_v, Cap)$ ;
15   Assign  $(u, v, w)$  to  $T_{best\_place}$ ;
16 // Step 2: Assign Remaining Nodes
17 for  $v \in V$  where  $v \notin C$  do
18    $best\_place \leftarrow \text{FindBestFit}(D_v, T, V_{avg}, R_v, Cap)$ ;
19   Assign  $v$  to  $T_{best\_place}$ ;
20 return  $T$ 

```

all 3-cliques are evenly distributed across different bins to achieve workload balancing. Secondly, after all cliques are processed, the remaining nodes are assigned to the most suitable bin to minimize the K-hop degree among all bins (Algorithm 2, Lines 17–19).

C. Clique-centric Memory Access Optimization

Higher-order GNNs are memory-bound due to the expanded aggregation field, and optimizing the off-chip memory access is necessary to improve performance and reduce energy consumption. We propose an analytical model based on the clique-centric computation to determine the adequate tiling size for each type of data: input (i.e., vertex features), intermediate results (i.e., the aggregation results of 3-clique), output (i.e., updated partial results of a specific hop), and weights.

We assume the on-chip memory capacity for input, IR, output, and weights as Mem_{in} , Mem_{IR} , Mem_{out} , and Mem_W . The objective of our memory optimization model is to find an adequate allocation of on-chip buffers among different data types, minimizing the total DRAM access. The on-chip memory size is constrained by the tile size T and the K-hop degree D_K .

DRAM Access Modeling for Input Graphs. Graph data needs to be tiled given the limited on-chip buffer size. The DRAM access of graph metadata in $G(V, E)$ and features can be modeled by:

$$DRAM_{in} = \left\lceil \frac{4 \times dim_F \times D_k}{T_{in}} \right\rceil \times T_{in} \quad (6)$$

here, dim_F represents the feature length using 32-bit floating-point precision (i.e., FP32). As such, each data point is 4 bytes.

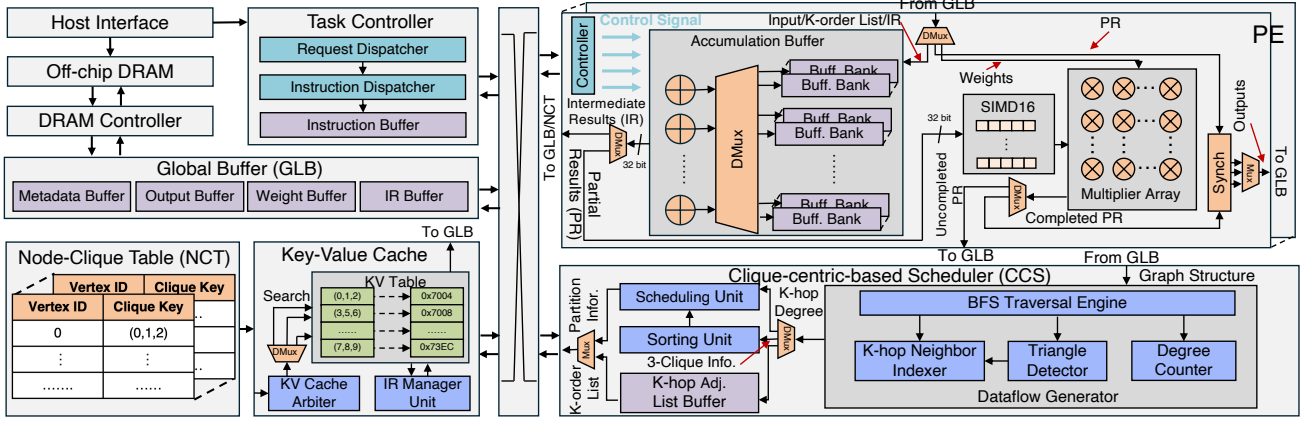


Fig. 7. The overview of the proposed architecture design.

DRAM Access Modeling for Intermediate Results. Since the redundancy elimination relies on the reuse of intermediate data, the tile size selection of intermediate data should be carefully modeled. To formulate the related DRAM access, the total number of IR (C_{IR}) and the total number of 3-cliques (Num_{clique}) should be considered. Therefore, the total number of DRAM accesses for intermediate results can be formulated as:

$$DRAM_{IR} = \left\lceil \frac{4 \times dim_F \times (C_{IR} + Num_{clique})}{T_{IR}} \right\rceil \times T_{IR} \quad (7)$$

DRAM Access Modeling for Output. The output buffer is used to store the partial results of aggregation and update. When all the hop aggregations are completed, these partial results will be read and concatenated as the final output. Assuming the weight size is dim_W , the output DRAM access latency can be calculated by:

$$DRAM_{out} = \left\lceil \frac{2 \times 4 \times K \times dim_W \times V}{T_{out}} \right\rceil \times T_{out} \quad (8)$$

DRAM Access Modeling for Weights. Unlike traditional GNNs, the same weight matrix is shared among vertices that are at the same hop distance from a source node. Consequently, the total memory consumption of higher-order GNN weights ($DRAM_W$) is formulated as:

$$DRAM_W = \left\lceil \frac{4 \times K \times dim_F \times dim_W}{T_W} \right\rceil \times T_W \quad (9)$$

Based on the above formulations, with the constraint of:

$$T_{in} + T_{IR} + T_{out} + T_W \leq Total_{Buffer} \quad (10)$$

The objective of the DRAM access optimization can be formulated as follows:

$$(T_{in}, T_{IR}, T_{out}, T_W) = \min(DRAM_{in} + DRAM_{IR} + DRAM_{out} + DRAM_W) \quad (11)$$

IV. PROPOSED ARCHITECTURE DESIGN

While a considerable number of accelerator architectures have been proposed for GNN models, they mainly target neighborhood aggregation. In this work, we target the reuse

of both neighborhood and path-overlapping aggregations by leveraging a clique-centric computation model. This approach necessitates the identification and reuse of 3-clique structures at runtime. In addition, higher-order GNNs perform parallel graph traversals to reach distant nodes several hops away, posing additional storage and computation demands. Current GPUs handle higher-order GNNs by processing the powers of the adjacency matrix and their aggregations in a parallel or sequential manner, where redundant computation and memory access are completely overlooked. The wide variability in sparsity across datasets, along with the increasing density of higher-order adjacency matrices, complicates abstraction into Sparse-Dense Matrix Multiplication (SpMM) or General Matrix Multiplication (GEMM) operations.

A. Architecture Overview

We propose a new accelerator to efficiently support clique-centric computation for higher-order GNNs. First, although adjacency matrices have been widely used in graph applications to abstract graph connectivity for matrix computations, computing and storing higher powers of the adjacency matrix is costly. Second, identifying and reusing 3-clique structures introduces new challenges in handling parallelism, workload balancing, and hardware support (e.g., KV cache). Figure 7 illustrates the overall architecture of the proposed accelerator. MOSAIC comprises the following key components: a multi-bank global buffer (GLB), a task controller, a clique-centric scheduler (CCS), a KV cache design with an associated search table, and a processing element (PE) array. The accelerator connects to external DRAM via a DRAM controller, while the PE array communicates with the GLB through a crossbar network. Specifically, the GLB stores inputs, outputs (e.g., partial sums and final results), weights, and intermediate results (e.g., cliques). The control unit oversees system operations and interfaces with the host through a request dispatcher. The dispatcher compiles incoming requests into instructions, which are forwarded to the instruction buffers. The instruction dispatcher monitors instruction execution and completion across the system, generates addresses for the instruction buffers, and routes instructions to the CCS and PE array.

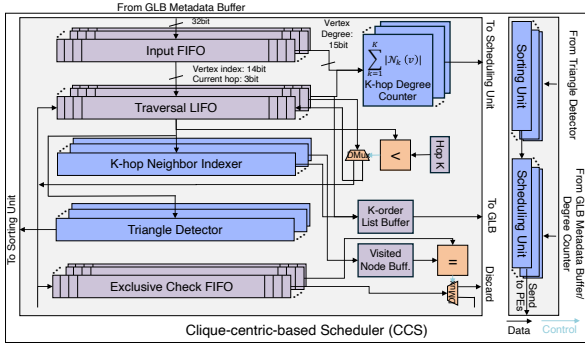


Fig. 8. Proposed clique-centric-based scheduler design.

B. Proposed Microarchitecture

The proposed microarchitecture features several unique designs to address the memory and computation challenges in higher-order GNNs. The proposed microarchitecture can directly process the raw graph data and generate a K-order list to record the higher-order graph connectivity. Moreover, we retrieve cliques from the CCS engine that implements a hardware-based triangle detection algorithm to dynamically identify 3-clique structures and reuse them through the KV cache. Lastly, we propose a PE architecture that supports clique reuse to eliminate both neighborhood and path-overlapping redundancy.

Specifically, as illustrated in Figure 7, the PE microarchitecture comprises two primary components: an accumulation buffer array and a multiplier array, corresponding to the aggregation and update phases of higher-order GNNs. When loading metadata from the GLB, the accumulation buffer array performs node feature aggregation under the guidance of the K-order list (which records higher-order graph connectivity) provided by the CCS engine. Next, the node-clique table (NCT) reads the identified 3-clique structures and fetches intermediate results to eliminate redundant computations. Subsequently, the multiplier array updates vertex features by processing 16-lane segments in parallel via a SIMD16 unit. Partially updated hop features are returned to the GLB as incomplete partial results, while a synchronization unit concatenates completed partial results from K-hop aggregation. The proposed aggregation and update units are integrated within the same PE architecture, thereby reducing write and read operations to the GLB.

Key-Value Cache. Due to the rich reuse opportunities of intermediate results, a KV Cache is managed to store all clique and IR data addresses, eliminating redundant computations and off-chip memory accesses. The proposed KV Cache is implemented as a fast, on-chip lookup table with a tag-based indexing mechanism, enabling low-latency retrieval. It provides two interfaces for pulling data from or pushing data to the distributed store. The pull interface leverages a hash-based indexing strategy to minimize lookup latency, while the push interface supports seamless insertion and eviction of data, maintaining effective cache behavior through a new replacement policy controlled by the KV Cache arbiter. Due to the irregular connectivity of graph nodes, different IRs are used at different time steps. Meanwhile, the limited GLB

size prevents all IRs from being stored on-chip, resulting in additional off-chip memory accesses.

In this work, we observe that the access frequency of IRs can be statistically estimated using the K-hop degree, providing insights into the IR replacement policy. Specifically, the IR management unit dynamically tracks the potential usage frequency of each IR, decrementing it after every access. Once the counter reaches zero, the IR is permanently discarded. To optimize on-chip storage, IRs with higher potential usage frequencies are prioritized, while those with lower frequencies are offloaded to off-chip memory when the on-chip buffer reaches capacity.

Node-clique Table. While the KV Cache employs a pull interface with a hash-based indexing strategy, it still falls short in searching for the stored address of an IR via a single node index because the KV Cache key is a list of three node indices. To address this, we propose a node-clique table to facilitate efficient searching of IRs using a single node index. By searching the node index, we can quickly retrieve the clique key of the KV Cache, thereby reducing the search complexity within the KV Cache.

C. Proposed Clique-centric-based Scheduler

Figure 8 illustrates the design of the proposed clique-centric-based scheduler (CCS) for graph traversal and partitioning. The system begins with *Input FIFOs* that receive and buffer graph nodes from the *Graph Metadata Buffer* prior to processing. A dedicated Traversal Unit subsequently dequeues vertices from depth-aware *Traversal LIFOs*, which maintain search hierarchy through stack-based management of traversal depth. Each entry in the *Traversal LIFOs* contains two key fields: a 14-bit vertex index (supporting graphs with up to 16,384 nodes) and a 3-bit current hop counter (enabling depth tracking up to 8 hops). This configuration accommodates higher-order GNN requirements whose traversal depth seldom exceeds 5 hops. During traversal, the *K-hop Neighbor Indexer* dynamically catalogs neighbors in the *K-order list Buffer*, while a 15-bit *K-hop Degree Counter* tracks neighborhood expansion magnitudes, sized to accommodate degrees up to twice the vertex count as empirically observed. A comparator-controlled loop ($<$) terminates traversal at preset hop limits, while the *Triangle Detector* identifies triangular relationships among 1-hop neighbors. To prevent redundant visits, the *Exclusive Check FIFO* verifies whether a node has been processed using the *Visited Node Buffer*, avoiding duplications (leading to “Discard”). In the final stage, the *Sorting Unit* prioritizes nodes forming 3-cliques, forwarding them to the **Scheduling Unit**, which assigns them to Processing Elements (PEs). Notably, the node-clique tables are constructed as well.

D. Execution Flow for Clique-centric Computation

At the scheduling stage, 3-cliques have been detected and stored, and the next challenge is to reuse the clique, eliminating redundant computations during K-hop aggregation. To better illustrate the execution flow of our proposed accelerator, we use the example as depicted in Figure 9. As mentioned,

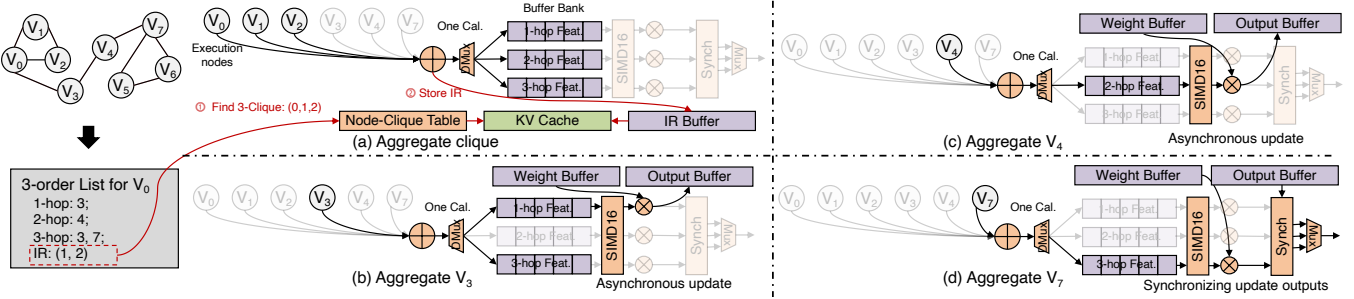


Fig. 9. A walkthrough example of our proposed workflow for clique-centric computation at V_0 .

TABLE II
DIMENSIONS AND HETEROPHILY OF GRAPH DATASETS.

Dataset	Vertices	Edges	Density	Heterophily
Cora (CR)	2,708	10,556	2.88×10^{-3}	Low
CiteSeer (CS)	3,312	9,104	1.66×10^{-3}	Low
PubMed (PM)	19,717	88,648	4.56×10^{-4}	Low
Reddit (RD)	232,965	114,615,892	4.22×10^{-3}	Low
YouTube (YB)	1,138,499	5,980,886	9.23×10^{-6}	Low
ogbn-products (OS)	2,449,029	61,859,140	2.06×10^{-5}	Low
ogbn-papers100M (OM)	111,059,956	1,615,685,872	2.62×10^{-7}	Low
OmniPath (OP)	16,325	289,134	2.17×10^{-3}	High
HumanNet (HN)	18,591	2,250,780	6.51×10^{-3}	High
Cornell (CN)	183	298	8.95×10^{-3}	High
Texas (TX)	183	309	1.86×10^{-2}	High
Wisconsin (WC)	251	499	1.59×10^{-2}	High
Actor (AC)	7,600	30,019	1.04×10^{-3}	High

the execution flow aims to reuse the intermediate results of 3-cliques for K-hop aggregation.

We use node V_0 as an example to illustrate its K-hop aggregation and update phases in the proposed PE architecture, as shown in Figure 9. The 3-order list of node V_0 will be loaded, which records its connectivity to K-hop neighbors (e.g., 1-hop, 2-hop, and 3-hop) and IR information. The IR information (i.e., IR (1,2)) will be used to search the Node-clique Table to find if the aggregated results of the clique have been computed. If the clique is not found in the node-clique table, the clique (i.e., V_0, V_1 , and V_2) will be computed through the accumulation unit. The intermediate result will be written to banked output buffers (e.g., 1-hop, 2-hop, 3-hop bank buffers), and KV cache. In the meantime, the Node-clique table will be updated. This provides two benefits. Firstly, for V_0 's K-hop aggregation, the IR will be directly reused. For example, in 2-hop aggregation, as shown in Figure 9(c), node V_4 will be loaded to the accumulation unit and can be directly added to IR. This removes redundant computations in K-hop aggregation. Secondly, when nodes V_1 , and V_2 perform K-hop aggregation, the IR will be reused as well. This eliminates repeated computations for neighborhoods. When the K-hop aggregation is done for each vertex, the result will be sent to the SIMD16 unit for GNN update.

V. EVALUATION

A. Experiment Setup

Environment Setup. To evaluate MOSAIC's performance, we conduct experiments using a cycle-accurate C++ simulator following methodologies that meticulously replicate every module within the higher-order GNN pipeline, ensuring

TABLE III
NODE CLASSIFICATION ACCURACY (%) OF TRADITIONAL GNN AND HIGHER-ORDER GNN MODELS.

Data set	Traditional GNNs			Higher-order GNNs		
	GCN	GraphSAGE	GAT	MixHop	GAT-JK	H2GCN
CR	87.27 ± 1.24	85.40 ± 1.22	85.34 ± 0.88	85.93 ± 1.09	84.33 ± 1.05	87.75 ± 0.88
CS	76.06 ± 1.13	73.90 ± 1.69	73.19 ± 1.79	74.99 ± 1.81	71.74 ± 1.95	75.95 ± 1.24
PM	87.38 ± 0.37	88.71 ± 0.68	87.13 ± 0.54	89.06 ± 0.48	85.56 ± 0.57	89.44 ± 0.46
OM	57.46 ± 0.25	58.00 ± 0.56	56.97 ± 0.26	62.46 ± 0.22	60.18 ± 0.22	63.44 ± 0.26
CN	47.89 ± 7.88	49.89 ± 6.84	46.05 ± 8.43	74.47 ± 4.71	50.53 ± 8.55	78.16 ± 6.97
TX	50.26 ± 3.21	40.75 ± 4.82	43.68 ± 5.79	84.21 ± 6.66	41.58 ± 5.24	86.58 ± 4.47
WC	45.49 ± 6.97	46.18 ± 4.90	44.12 ± 6.34	85.69 ± 3.93	46.67 ± 8.03	85.88 ± 4.19
AC	27.18 ± 0.98	28.91 ± 1.13	26.36 ± 1.11	32.70 ± 1.23	26.68 ± 0.82	35.33 ± 1.06

Values are colored according to their ranking: **best in red, second-best in blue.**

a comprehensive representation of the system architecture. Additionally, we integrate our simulator with Ramulator [73], configured as high-bandwidth memory (HBM) [74] with a total bandwidth of 256 GB/s to model off-chip storage. The HBM consists of 8 channels, each with a 128-bit bus, allowing the simulation to capture potential channel conflicts due to irregular memory access. MOSAIC is configured on an 8×8 processing element (PE) array with a 64 MB global buffer and an interconnect bandwidth of 400 GB/s per PE. The PE clock frequency is set to 1 GHz by default. Additionally, each PE includes a 4×4 multiplier array and 32 SIMD cores. To validate the simulator's accuracy, we compare the timing behavior of its simulated modules against synthesized RTL designs. These RTL designs, implemented in SystemVerilog HDL, are synthesized using Synopsys Design Compiler with the TSMC 32 nm standard cell library. For numerical computations, we use IEEE 754 single-precision floating-point representation. For estimating the area and power of on-chip buffers, we use Cacti 6.0 with 32 nm technology [75]. For GPU results, we conduct our experiments on a server with an Intel® Xeon® Platinum 8362 CPU @ 2.80 GHz and an NVIDIA A100 80 GB GPU [76].

Baselines and GNN models. We compare the performance and energy efficiency of MOSAIC with three state-of-the-art GNN baseline designs: Betty [2], ReGNN [3], and I-GCN [1]. For a fair comparison, all baseline designs are scaled to match MOSAIC in terms of the number of multiply-and-accumulate (MAC) units and off-chip/on-chip bandwidth. In our implementation, we faithfully replicate I-GCN's islandization strategy to streamline the handling of dense submatrices. For ReGNN, we adopt the framework's redundancy elimination methodology, as outlined in the design specifications [3], to minimize unnecessary computations and inter-process commu-

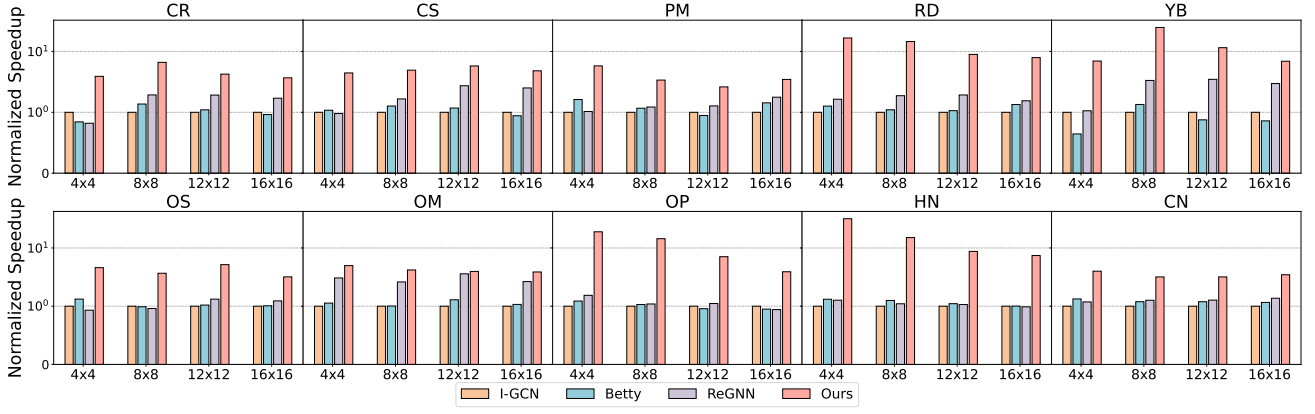


Fig. 10. Normalized speedup of baseline designs and MOSAIC over I-GCN across 10 datasets.

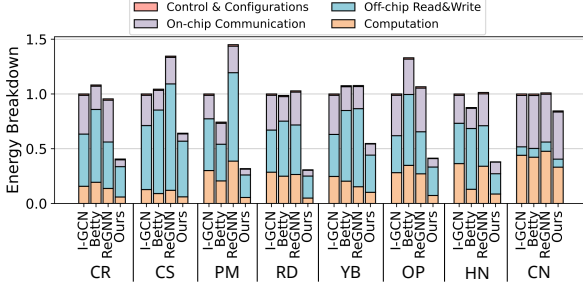


Fig. 11. Energy efficiency breakdown across 8 datasets.

nication. As for Betty, we simulate the redundancy-embedded graph partitioning in higher-order GNNs. It should be noted that all optimizations are only for 1-hop GNNs. For comparative analysis, we employ the MixHop [26] architecture as a standard higher-order GNN model for all designs, where we define the hop as 2. To benchmark the performance of higher-order GNNs, we include methods utilizing higher-order (e.g., MixHop, GAT-JK [58], and H2GCN [52]) against a comprehensive suite of baselines representing key strategies in graph representation learning. Specifically, we employed three well-established GNN models: Graph Convolutional Network (GCN) [4], [77], GraphSAGE [6], and Graph Attention Network (GAT) [5]. Specifically, we use identical hyperparameters for the GNN models, such as 200 epochs, a learning rate of 0.001, weight decay of $1e^{-4}$, and a hidden dimension size of 128.

Datasets. We benchmark our design against the baselines for higher-order GNN on diverse datasets [6], [78]–[82], as detailed in Table II. The selected datasets span scientific, medical, social, biological, and web networks, encompassing both homophilic and heterophilic graphs with varying densities and feature dimensions. Notably, Cornell is included as a representative of highly heterophilic graphs to assess performance under challenging conditions.

B. Performance Evaluation

To evaluate the performance of MOSAIC, we compared MOSAIC with I-GCN, Betty, and ReGNN for higher-order GNNs, measured by the total number of execution cycles. For a fair comparison, all accelerators are configured with an 8×8 PE array and a 1 KB feature size. As shown in Figure 10, MOSAIC outperforms I-GCN, Betty, and ReGNN

by $9.48\times$, $7.46\times$, and $5.49\times$, respectively. The performance improvement comes from reduced DRAM access and alleviated redundant computations. Specifically, in large graphs (e.g., OS and OM), the high sparsity of these graphs inherently exhibits fewer neighborhood and path-overlapping redundancies. Thus, the effectiveness of all the optimization techniques is limited by the sparse graph connectivity. Moreover, MOSAIC demonstrates better performance in social networks (e.g., RD and YB) and biological networks (e.g., OP and HN), as these networks naturally form clusters or ‘communities’ where nodes are highly clustered in several regions (e.g., approximately 20% of nodes contribute to 80% redundancy). As compared to prior works, MOSAIC presents that clique-centric computations can eliminate redundant computations while balancing the workload across update and aggregation phases. However, I-GCN, Betty, and ReGNN have limited applicability to address the path-overlapping redundancy. Furthermore, Betty repurposes METIS partitioning [2], [83] to identify mutually shared nodes to optimize data reuse, where the path-overlapping issue is not addressed. ReGNN utilizes redundancy-free message passing to identify redundant sets to avoid redundant one-hop aggregation, and, similarly, the path-overlapping redundancy cannot be fully exploited by identifying the graph connectivity of the immediate neighborhood.

Model-level accuracy. We select six representative deep (e.g., GCN, GraphSAGE, GAT) and higher-order (e.g., MixHop, GAT-JK, H2GCN) GNNs and compare model accuracy across different graphs. As shown in Table III, MixHop achieves average accuracy improvements of 13.82%, 14.72%, and 15.83% over GCN, GraphSAGE, and GAT, respectively. Specifically, MixHop showcases its strength and adaptability on the more challenging heterophilic datasets (e.g., CN, TX, WC, AC), which present the enhanced practicality of higher-order GNNs. Our proposed acceleration framework significantly reduces the computation overhead of running higher-order GNNs, thereby enabling their broader adoption across real-world applications.

C. Energy Efficiency

Figure 11 details the energy breakdown analysis, including computation, on-chip communication, off-chip data Read&Write, and cost of control & configuration, where NoC energy is modeled using DSENT [84]. MOSAIC shows an

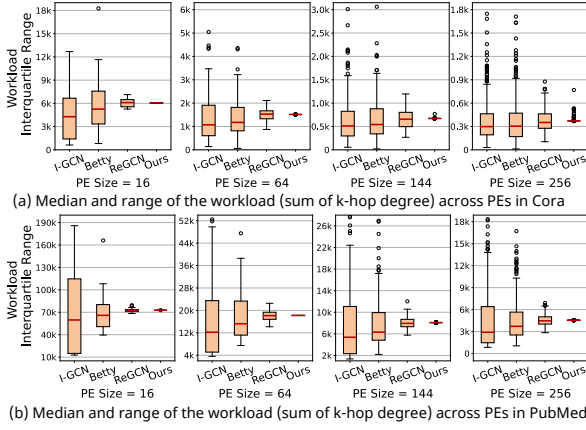


Fig. 12. The workload evaluation: workload interquartile range of I-GCN, Betty, ReGNN, and MOSAIC with varying PE size.

average energy reduction of 51.78%, 52.53%, and 56.84% compared with I-GCN, Betty, and ReGNN, respectively. Specifically, MOSAIC reduces average DRAM access and computation energy consumption by 33.57% and 62.89% compared with I-GCN, respectively. The energy reduction comes from enhanced IR reuse by the proposed KV cache, which reduces read-write operations to DRAM. In addition, reusing the IRs of 3-cliques can reduce energy consumption of computation units. In HN and OP, MOSAIC achieves a higher reduction in DRAM and computation due to a large number of cliques. The increased IR reuse also leads to reduced on-chip communication, where the repeated fetches of vertices and features are avoided.

D. Scalability Study

To analyze the scalability of MOSAIC, the PE array dimensions are scaled from 4×4 to 16×16 , resulting in a total MAC count ranging from 256 to 4096. As shown in Figure 10, MOSAIC has superior scalability compared to baseline architectures, achieving average speedups of $7.67\times$, $6.73\times$, and $4.88\times$ over I-GCN, Betty, and ReGNN across all datasets and PE array sizes, respectively. MOSAIC shows better scalability primarily due to two reasons: balanced workload and reduced communication. As compared to edge or vertex-centric computation, MOSAIC partitions the workload based on clique structure and K-hop degree information, which essentially eliminates the communication between cliques and balances the K-hop aggregation. As shown in Figure 10, where lines are the average speedup of MOSAIC over I-GCN, Betty, and ReGNN, the performance gains of MOSAIC diminish slightly when the PE size increases to 16×16 . This is because the long-distance communication is inevitable in most of the graph datasets, which could significantly affect the performance when the PE size scales.

E. Workload Distribution

Our design aims to achieve workload balance across both higher-order GNN execution phases: the aggregation phase, which is affected by K-hop vertex degrees, and the update phase, which depends on the number of vertices. To quantify aggregation workload distribution, we employ the Interquartile

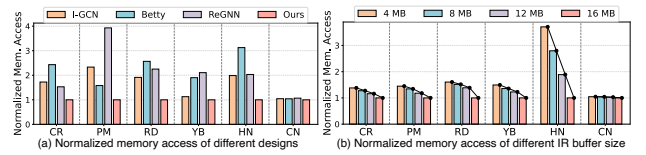


Fig. 13. Normalized DRAM access of (a) Different designs and (b) Different IR buffer sizes.

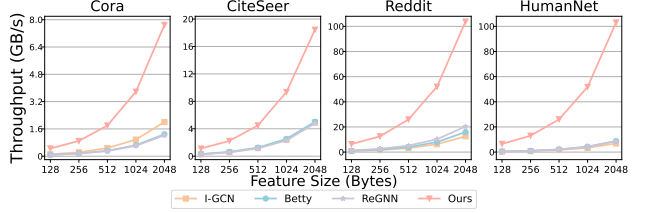


Fig. 14. Throughput comparison across different data sizes.

Range (IQR), a statistical measure of data dispersion. By design, our approach ensures that all PEs handle equivalent aggregation workloads. Due to space constraints, we present results only for the Cora and PubMed datasets.

As shown in Figure 12, MOSAIC significantly outperforms baseline designs. For example, on the Cora dataset, MOSAIC achieves an average IQR of 0.25, compared to values of 1841.8 (I-GCN), 1521.3 (Betty), and 440.0 (ReGNN). This stark contrast underscores MOSAIC's superior workload balancing, which directly translates to higher resource utilization. In contrast, baselines focus solely on equalizing vertex distribution, neglecting aggregation workload balance. Notably, I-GCN exacerbates imbalance by clustering high-connectivity vertices (1-hop) into a few partitions via a BFS strategy. This creates densely connected regions in early-formed partitions while leaving others sparser. Thus, the workload imbalance issue of I-GCN is more severe.

F. Throughput Analysis

As shown in Figure 14, we vary feature size from 128 B to 2 KB to cover both small and large-scale scenarios, and record the simulation cycles and data transfer volume. By dividing the data transfer volume by the number of simulation cycles, we calculate the throughput by considering clock frequency, where the throughput is reported in Gigabytes per second. On average, MOSAIC achieves $4.67\times$, $4.81\times$, and $4.95\times$ higher throughput than I-GCN, Betty, and ReGNN, demonstrating efficient bandwidth utilization and high parallelism. Unlike prior approaches constrained by the adjacency matrix in 1-hop aggregation, MOSAIC benefits from tremendous redundancy elimination and off-chip memory access reduction since the cliques are universal in higher-order GNNs.

G. Off-chip Memory Access

To evaluate memory efficiency, we measure off-chip memory access, as shown in Figure 13(a). MOSAIC demonstrates a significant reduction in off-chip memory accesses compared to the baselines, achieving average improvements of $1.69\times$, $2.11\times$, and $2.15\times$, respectively. While prior works use tiling and workload scheduling strategies, their performance is subject to sparsity variations, which often result in inconsistent memory access patterns across datasets. In contrast, MOSAIC

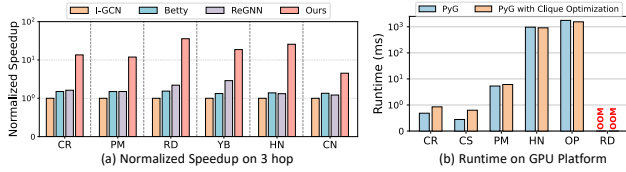


Fig. 15. (a) Normalized speedup over I-GCN on 3 hops. (b) Runtime comparison between vanilla PyG implementation and PyG with clique optimization on GPUs.

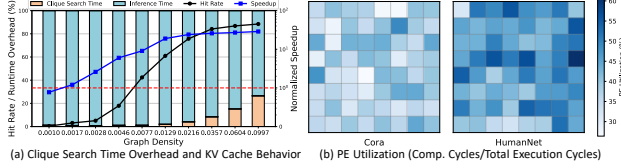


Fig. 16. (a) Clique search time overhead and Cache Behavior across different densities. (b) PE utilization analysis.

consistently outperforms the baselines by reusing intermediate results without redundantly fetching raw data. I-GCN reduces redundant memory accesses by co-locating highly connected vertices with their neighbors, thereby enhancing data locality and better handling irregular access patterns, outperforming the other two accelerator designs in this aspect. ReGNN, while mitigating off-chip memory access through efficient caching of redundant sets, only eliminates neighborhood redundancy. This makes ReGNN inefficient in optimizing off-chip memory access for higher-order GNNs. On the CN dataset, MOSAIC’s off-chip memory reduction is marginal due to the dataset’s small scale. In contrast, on the HN dataset, MOSAIC achieves the highest off-chip memory reduction, benefiting from the graph’s high connectivity.

H. Sensitivity and Ablation Study

Off-chip Memory Access on Different IR Buffer Size. We conduct a sensitivity study to evaluate the impact of IR buffer size on off-chip memory access. As shown in Figure 13(b), increasing the IR buffer size leads to an average reduction of 43.80% in off-chip memory access. For the CN dataset, due to its small scale, even a small IR buffer can accommodate all intermediate results, resulting in minimal differences across buffer sizes. In contrast, for the HN dataset, the large variation in off-chip memory access is attributed to the high connectivity of the graph (e.g., 68.19% of nodes are involved in non-overlapping 3-clique constructions).

Comparison on GPU Platform. To evaluate the effectiveness of clique-centric optimization on general-purpose hardware, we implement the proposed clique-centric dataflow with K-hop degree partitioning using PyG [33] and compare it against vanilla PyG. As shown in Figure 15(b), clique optimization increases runtime on homophilic graphs (CR, CS, PM). Specifically, while clique-centric computation eliminates redundant aggregations, this is realized by duplicating cliques via `torch.unique`, which internally requires an $\mathcal{O}(E \log E)$ reordering of the full edge set and serializes the preceding index remapping and feature matrix construction into a sequential chain of kernel launches, destroying the parallelism that GPU execution relies on. Furthermore, clique

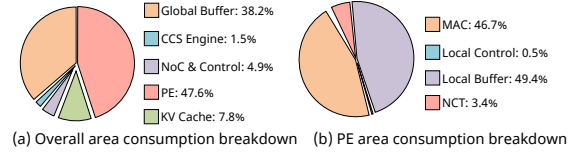


Fig. 17. Area breakdown analysis.

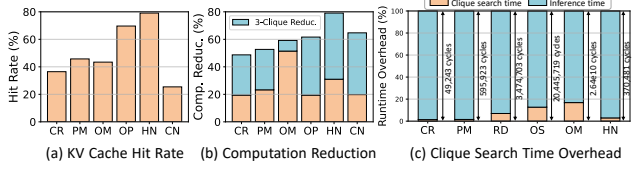


Fig. 18. (a) KV cache hit rate, (b) computation reduction, and (c) Clique search time.

IR retrieval and reuse exhibit irregular, non-coalesced memory access patterns. In addition, adding clique IR into the feature matrix during the gather step inside `propagate` introduces control flow divergence that undermines the GPU’s optimized SIMT execution. While moderate speedups are observed on other graphs, the benefits remain limited without clique-centric scheduling and dedicated KV-cache support.

Performance on Larger Hop Distance. We further evaluate the performance of our proposed technique when increasing the receptive field of K-hop aggregation. Specifically, we increase the hop distance to 3 hops. As illustrated in Figure 15(a), MOSAIC outperforms 18.35 $\times$, 12.79 $\times$, and 10.24 $\times$ over the I-GCN, Betty, and ReGNN in 4 $\times$ 4 PE configuration, respectively. As aforementioned, the performance gains stem from the superiority of workload balance and IR reuse design that are designed specifically for higher-order GNNs. Notably, MOSAIC’s performance gains increase in 3-hop aggregation due to increased path-overlapping redundancy.

Cache Behavior across Different Graph Densities. To evaluate the effect of graph densities, we artificially generate synthetic graphs with different densities with a fixed number of vertices (i.e., 2,000), where the clique structure is much fewer than typical homophilic or heterophilic graphs. We compare cache hit rate and corresponding speedup over I-GCN with the 2-hop higher-order GNNs in a 4 $\times$ 4 PE configuration, as shown in Figure 16(a). As graph density increases, both the KV cache hit rate and the resulting performance speedup improve. When clique reuse is negligible, resulting in a KV cache hit rate of only 0.41%, MOSAIC cannot outperform I-GCN. However, once a moderate level of clique reuse is available, the KV cache can effectively exploit reusable IR, enabling MOSAIC to consistently deliver performance improvements.

PE Utilization. Figure 16(b) shows the PE utilization across an 8 $\times$ 8 PE array using CR and HN. Despite their substantially different clique densities, both graphs maintain comparable PE utilization with prior work [36], where HN is marginally higher than CR by 8.81%. This small gap demonstrates that MOSAIC’s clique-centric partitioning consistently balances workloads across diverse graph structures, indicating that the reported performance gains stem from both better data locality and workload balancing.

I. Overhead Analysis

Area Analysis. Figure 17(a) shows the breakdown of the overall area of the proposed accelerator. This includes the PE, global buffer, CCS engine, networks on chip (NoC), and logic components for control and configuration. The PE array accounts for 47.6% of the total chip area, and the on-chip global buffer consumes 38.2% of the total area. The NoC and control together use 4.9% of the chip area, and the KV cache takes 7.8%. The CCS engine's area consumption is negligible at 1.5% of the total chip area. Figure 17(b) provides a breakdown of the total PE area, where the MAC units account for 46.7%. In contrast, the Local buffer and the NCT account for 49.4% and 3.4%, respectively. The PE reconfigurable local control logic has a negligible area consumption of 0.5% of the total PE area.

Clique Searching Overhead. As shown in Figure 16(a), the search time accounts for only 5.9% of the total execution time on average. However, its overhead becomes non-negligible when the graph density exceeds 5%. On the other hand, the benefits of KV cache reuse become more significant as density increases due to the higher cache hit rate. This helps maintain the overall performance speedup at a similar level despite the increased preprocessing cost. Furthermore, since the preprocessing results are reused across multiple inference runs or training epochs, the associated cost can be effectively amortized over repeated executions, especially for higher-order GNNs with large receptive fields, i.e., larger hop (K).

Figure 18(c) shows the overhead of finding cliques. The search time is negligible (7.06%) as compared to the overall runtime. This low cost highlights the practicality of incorporating clique structures into higher-order GNNs.

J. KV Cache Performance Analysis

KV cache performance and computation reduction are subject to variation in graph structure.

Cache Hit Rate. Figure 18(a) shows the KV cache hit rate with a fixed 16 MB buffer. The average hit rate is 49.97%. HN (79.09%) achieves the highest hit rate owing to its highly connected structure. KV hit rate depends on the graph connectivity, where denser graphs show higher path-overlapping. This leads to higher reuse and a higher hit rate of 3-cliques.

Computation Reduction. To evaluate the impact of IR reuse, we conduct a study to measure the computation reduction of MOSAIC. As shown in Figure 18(b), MOSAIC reduces computation by an average of 61.01%. The trend closely aligns with that of the hit rate, as both are strongly influenced by the graph structure. Notably, datasets with higher computation reduction exhibit correspondingly significant improvements. Similarly, the HN dataset shows the greatest reduction due to its denser connectivity. Specifically, 54.60% computation reduction is introduced by 3-cliques.

VI. CONCLUSION

This paper proposes MOSAIC, a clique-centric accelerator designed to decouple the computation and memory overheads of higher-order aggregation from the expanded receptive field

in higher-order GNNs. The core insight is that clique graphs serve as the fundamental structures capturing overlapping paths in distant feature aggregation. Building on this idea, MOSAIC delivers performance and latency comparable to traditional GNNs while significantly improving efficiency. First, we introduce a clique-centric dataflow that eliminates redundant computation and memory access arising from path overlaps during simultaneous multi-hop aggregations. Second, we present a K-hop degree partitioning algorithm that balances workloads by leveraging degree information from extended receptive fields. Third, we propose a novel data representation that encodes multi-distance graph connectivity without requiring separate adjacency matrices. Lastly, we design a specialized accelerator architecture that enables clique reuse, avoiding repeated computation in concurrent aggregations. Experiment results show that MOSAIC achieves average speedups of 7.67 $\times$, 6.73 $\times$, and 4.88 $\times$ and energy reduction of 51.78%, 52.53%, and 56.84% over I-GCN [1], Betty [2], and ReGNN [3] in terms of support for higher-order GNNs, respectively.

ACKNOWLEDGMENT

This work was partially supported by the U.S. National Science Foundation under CAREER Award CCF-2441973 and Awards CCF-2551106, CCF-2551105, CNS-2321224, and CCF-2131946, and the U.S. Department of Energy under grant DE-SC0024428. We sincerely thank the reviewers for their valuable comments.

REFERENCES

- [1] Tong Geng, Chunshu Wu, Yongan Zhang, Cheng Tan, Chenhao Xie, Haoran You, Martin Herbordt, Yingyan Lin, and Ang Li. I-GCN: A graph convolutional network accelerator with runtime locality enhancement through islandization. In *Proceedings of IEEE/ACM International Symposium on Microarchitecture (MICRO)*, pages 1051–1063. IEEE, 2021.
- [2] Shuangyan Yang, Minjia Zhang, Wenqian Dong, and Dong Li. Betty: Enabling large-scale GNN training with batch-level graph partitioning. In *Proceedings of ACM International Conference on Architectural Support for Programming Languages and Operating Systems (ASPLOS)*, pages 103–117, 2023.
- [3] Cen Chen, Kenli Li, Yangfan Li, and Xiaofeng Zou. ReGNN: A redundancy-eliminated graph neural networks accelerator. In *Proceedings of IEEE International Symposium on High-Performance Computer Architecture (HPCA)*, pages 429–443. IEEE, 2022.
- [4] Thomas N Kipf and Max Welling. Semi-supervised classification with graph convolutional networks. *arXiv preprint arXiv:1609.02907*, 2016.
- [5] Petar Velićković, Guillem Cucurull, Arantxa Csanova, Adriana Romero, Pietro Lio, and Yoshua Bengio. Graph attention networks. *arXiv preprint arXiv:1710.10903*, 2017.
- [6] Will Hamilton, Zitao Ying, and Jure Leskovec. Inductive representation learning on large graphs. In *Proceedings of Advances in Neural Information Processing Systems (NeurIPS)*, volume 30, 2017.
- [7] Jiarui Feng, Yixin Chen, Fuhai Li, Anindya Sarkar, and Muhan Zhang. How powerful are k-hop message passing graph neural networks. In *Proceedings of Advances in Neural Information Processing Systems (NeurIPS)*, pages 4776–4790, 2022.
- [8] Lingxiang Yin, Sanjay Gandham, Mingjie Lin, and Hao Zheng. Scale: A structure-centric accelerator for message passing graph neural networks. In *Proceedings of IEEE/ACM International Symposium on Microarchitecture (MICRO)*, pages 580–593. IEEE, 2024.
- [9] Sanjay Gandham, Lingxiang Yin, Hao Zheng, and Mingjie Lin. Saga: Sparsity-agnostic graph convolutional network acceleration with near-optimal workload balance. In *Proceedings of International Conference on Computer Aided Design (ICCAD)*, pages 1–9. IEEE, 2023.

- [10] Aravind Sankar, Yozen Liu, Jun Yu, and Neil Shah. Graph neural networks for friend ranking in large-scale social platforms. In *Proceedings of the Web Conference (WWW)*, pages 2535–2546, 2021.
- [11] Shengjie Min, Zhan Gao, Jing Peng, Liang Wang, Ke Qin, and Bo Fang. Stgsn—a spatiotemporal graph neural network framework for time-evolving social networks. *Knowledge-Based Systems*, page 106746, 2021.
- [12] Yi Wang, Feng Zhan, Cuiyu Huang, and Yiran Huang. Gbnss: A method based on graph neural networks (gnns) for global biological network similarity search. *Applied Sciences*, page 9844, 2024.
- [13] Giulia Muzio, Leslie O’Bray, and Karsten Borgwardt. Biological network analysis with deep learning. *Briefings in bioinformatics*, pages 1515–1530, 2021.
- [14] Xiao-Meng Zhang, Li Liang, Lin Liu, and Ming-Jing Tang. Graph neural networks and their current applications in bioinformatics. *Frontiers in genetics*, page 690049, 2021.
- [15] José Suárez-Varela, Paul Almasan, Miquel Ferriol-Galmés, Krzysztof Rusek, Fabien Geyer, Xiangli Cheng, Xiang Shi, Shihan Xiao, Franco Scarselli, Albert Cabellos-Aparicio, et al. Graph neural networks for communication networks: Context, use cases and opportunities. *IEEE network*, pages 146–153, 2022.
- [16] Yifei Shen, Jun Zhang, SH Song, and Khaled B Letaief. Graph neural networks for wireless communications: From theory to practice. *IEEE Transactions on Wireless Communications (TWC)*, pages 3554–3569, 2022.
- [17] Huiyuan Chen, Chin-Chia Michael Yeh, Fei Wang, and Hao Yang. Graph neural transport networks with non-local attentions for recommender systems. In *Proceedings of the ACM Web Conference (WWW)*, pages 1955–1964. ACM, 2022.
- [18] Wenqi Fan, Yao Ma, Qing Li, Yuan He, Eric Zhao, Jiliang Tang, and Dawei Yin. Graph neural networks for social recommendation. In *Proceedings of the World Wide Web Conference (WWW)*, pages 417–426, 2019.
- [19] Sitao Luan, Chenqing Hua, Qincheng Lu, Jiaqi Zhu, Mingde Zhao, Shuyuan Zhang, Xiao-Wen Chang, and Doina Precup. Revisiting heterophily for graph neural networks. In *Proceedings of Advances in Neural Information Processing Systems (NeurIPS)*, pages 1362–1375, 2022.
- [20] Xin Zheng, Yi Wang, Yixin Liu, Ming Li, Miao Zhang, Di Jin, Philip S Yu, and Shirui Pan. Graph neural networks for graphs with heterophily: A survey. *arXiv preprint arXiv:2202.07082*, 2022.
- [21] Junhong Lin, Xiaojie Guo, Shuaicheng Zhang, Dawei Zhou, Yada Zhu, and Julian Shun. When heterophily meets heterogeneity: New graph benchmarks and effective methods. *arXiv preprint arXiv:2407.10916*, 2024.
- [22] Christopher Morris, Martin Ritzert, Matthias Fey, William L Hamilton, Jan Eric Lenssen, Gaurav Rattan, and Martin Grohe. Weisfeiler and leman go neural: Higher-order graph neural networks. In *Proceedings of the AAAI Conference on Artificial Intelligence AAAI*, volume 33, pages 4602–4609, 2019.
- [23] Daniel Flam-Shepherd, Tony C Wu, Pascal Friederich, and Alan Aspuru-Guzik. Neural message passing on high order paths. *Machine Learning: Science and Technology*, 2(4):045009, 2021.
- [24] Kevin A Murgas, Emil Saucan, and Romeil Sandhu. Hypergraph geometry reflects higher-order dynamics in protein interaction networks. *Scientific Reports*, 12(1):20879, 2022.
- [25] Maciej Besta, Florian Scheidl, Lukas Gianinazzi, Grzegorz Kwasniewski, Shachar Klaiman, Jürgen Müller, and Torsten Hoefer. Demystifying higher-order graph neural networks. *arXiv preprint arXiv:2406.12841*, 2024.
- [26] Sami Abu-El-Hajja, Bryan Perozzi, Amol Kapoor, Nazanin Alipourfard, Kristina Lerman, Hrayr Harutyunyan, Greg Ver Steeg, and Aram Galstyan. Mixhop: Higher-order graph convolutional architectures via sparsified neighborhood mixing. In *International Conference on Machine Learning (ICML)*, pages 21–29. PMLR, 2019.
- [27] Zequn Sun, Chengming Wang, Wei Hu, Muhao Chen, Jian Dai, Wei Zhang, and Yuzhong Qu. Knowledge graph alignment network with gated multi-hop neighborhood aggregation. In *Proceedings of the AAAI Conference on Artificial Intelligence (AAAI)*, pages 222–229, 2020.
- [28] Claudio Gallicchio and Alessio Micheli. Fast and deep graph neural networks. In *Proceedings of the AAAI Conference on Artificial Intelligence (AAAI)*, volume 34, pages 3898–3905, 2020.
- [29] Guosheng Feng, Hongzhi Wang, and Chunnan Wang. Search for deep graph neural networks. *Information Sciences*, 649:119617, 2023.
- [30] Yanqiao Zhu, Weizhi Xu, Jinghao Zhang, Qiang Liu, Shu Wu, and Liang Wang. A survey on graph structure learning: Progress and opportunities. *arXiv preprint arXiv:2103.03036*, 2021.
- [31] Rex Ying, Ruining He, Kaifeng Chen, Pong Eksombatchai, William L Hamilton, and Jure Leskovec. Graph convolutional neural networks for web-scale recommender systems. In *Proceedings of the 24th ACM SIGKDD international conference on knowledge discovery & data mining (SIGKDD)*, pages 974–983, 2018.
- [32] Wanying Ding, Vinay K Chaudhri, Naren Chittar, and Krihshna Konakanchi. Jel: applying end-to-end neural entity linking in jpmorgan chase. In *Proceedings of the AAAI Conference on Artificial Intelligence (AAAI)*, volume 35, pages 15301–15308, 2021.
- [33] Matthias Fey and Jan Eric Lenssen. Fast graph representation learning with pytorch geometric. *arXiv preprint arXiv:1903.02428*, 2019.
- [34] Da Zheng, Chao Ma, Minjie Wang, Jinjing Zhou, Qidong Su, Xiang Song, Quan Gan, Zheng Zhang, and George Karypis. DistDGL: Distributed graph neural network training for billion-scale graphs. In *Proceedings of IEEE/ACM Workshop on Irregular Applications: Architectures and Algorithms (IA3)*, pages 36–44, 2020.
- [35] Gerasimos Gerogiannis, Sriram Ananthakrishnan, Josep Torrellas, and Ibrahim Hur. Hottiles: Accelerating spmm with heterogeneous accelerator architectures. In *Proceedings of IEEE International Symposium on High-Performance Computer Architecture (HPCA)*, pages 1012–1028. IEEE, 2024.
- [36] Tong Geng, Ang Li, Runbin Shi, Chunshu Wu, Tianqi Wang, Yanfei Li, Pouya Haghi, Antonino Tumeo, Shuai Che, Steve Reinhardt, and Martin C. Herbordt. AWB-GCN: A graph convolutional network accelerator with runtime workload rebalancing. In *Proceedings of IEEE/ACM International Symposium on Microarchitecture (MICRO)*, pages 922–936. IEEE, 2020.
- [37] Jiajun Li and Ahmed Louri. SGCNAX: A scalable graph convolutional neural network accelerator with workload balancing. In *IEEE Transactions on Parallel and Distributed Systems (TPDS)*, pages 2834–2845, 2022.
- [38] Gerasimos Gerogiannis, Serif Yesil, Damitha Lenadora, Dingyuan Cao, Charith Mendis, and Josep Torrellas. Spade: A flexible and scalable accelerator for spmm and sddmm. In *Proceedings of International Symposium on Computer Architecture (ISCA)*, pages 1–15. IEEE, 2023.
- [39] Zhiyao Li, Jiaxiang Li, Taijie Chen, Dimin Niu, Hongzhong Zheng, Yuan Xie, and Mingyu Gao. Spada: Accelerating sparse matrix multiplication with adaptive dataflow. In *Proceedings of ACM International Conference on Architectural Support for Programming Languages and Operating Systems (ASPLOS)*, pages 747–761. ACM, 2023.
- [40] Yifan Yang, Joel S Emer, and Daniel Sanchez. Trapezoid: A versatile accelerator for dense and sparse matrix multiplications. In *Proceedings of ACM/IEEE 51st Annual International Symposium on Computer Architecture (ISCA)*, pages 931–945. IEEE, 2024.
- [41] Minjie Yu Wang. Deep graph library: Towards efficient and scalable deep learning on graphs. In *Proceedings of ICLR workshop on representation learning on graphs and manifolds*, 2019.
- [42] Hongwu Peng, Xi Xie, Kaustubh Shivdikar, Md Amit Hasan, Jiahui Zhao, Shaoyi Huang, Omer Khan, David Kaeli, and Caiwen Ding. Maxk-gnn: Extremely fast gpu kernel design for accelerating graph neural networks training. In *Proceedings of ACM International Conference on Architectural Support for Programming Languages and Operating Systems (ASPLOS)*, pages 683–698, 2024.
- [43] Yuke Wang, Boyuan Feng, Gushu Li, Shuangchen Li, Lei Deng, Yuan Xie, and Yufei Ding. GNNAdvisor: An adaptive and efficient runtime system for GNNs acceleration on GPUs. In *Proceedings of USENIX Symposium on Operating Systems Design and Implementation (OSDI)*, pages 515–531. USENIX, 2021.
- [44] Haoyang Wang, Shengbing Zhang, Xiaoya Fan, Zhao Yang, and Meng Zhang. NDPGNN: A near-data processing architecture for GNN training and inference acceleration. *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems (TCAD)*, 43(11):3997–4008, 2024.
- [45] Amir Ghazizadeh Ahsaei, Lingxiang Yin, Shilin Tian, Fangzhou Ye, Fan Yao, and Hao Zheng. Rethinking tiling and dataflow for spmm acceleration: A graph transformation framework. In *Proceedings of the 58th IEEE/ACM International Symposium on Microarchitecture (MICRO)*, pages 1535–1548, 2025.
- [46] Fangzhou Ye, Shilin Tian, Amir Ghazizadeh Ahsaei, and Hao Zheng. Tensorprism: Rethinking sparse high-order tensor acceleration via co-occurrence graph. In *Proceedings of 2026 ACM/IEEE 53rd Annual*

- International Symposium on Computer Architecture (ISCA)*, pages 291–306. IEEE, 2026.
- [47] Yingxia Shao, Hongzheng Li, Xizhi Gu, Hongbo Yin, Yawen Li, Xupeng Miao, Wentao Zhang, Bin Cui, and Lei Chen. Distributed graph neural network training: A survey. *ACM Computing Surveys*, pages 1–39, 2024.
 - [48] Haiyang Lin, Mingyu Yan, Xiaochun Ye, Dongrui Fan, Shirui Pan, Wenguang Chen, and Yuan Xie. A comprehensive survey on distributed training of graph neural networks. *Proceedings of the IEEE*, pages 1572–1606, 2023.
 - [49] Fangzhou Ye, Lingxiang Yin, and Hao Zheng. Scaling graph neural network training via geometric optimization. In *Proceedings of 2026 IEEE International Symposium on High Performance Computer Architecture (HPCA)*, pages 1–15. IEEE, 2026.
 - [50] Rishov Sarkar, Stefan Abi-Karam, Yuqi He, Lakshmi Sathidevi, and Cong Hao. FlowGNN: A dataflow architecture for real-time workload-agnostic graph neural network inference. In *Proceedings of IEEE International Symposium on High-Performance Computer Architecture (HPCA)*, pages 1099–1112. IEEE, 2023.
 - [51] Deli Chen, Yankai Lin, Wei Li, Peng Li, Jie Zhou, and Xu Sun. Measuring and relieving the over-smoothing problem for graph neural networks from the topological view. In *Proceedings of the Annual Conference on Artificial Intelligence (AAAI)*, volume 34, pages 3438–3445, 2020.
 - [52] Jiong Zhu, Yujun Yan, Lingxiao Zhao, Mark Heimann, Leman Akoglu, and Danai Koutra. Beyond homophily in graph neural networks: Current limitations and effective designs. In *Proceedings of Advances in Neural Information Processing Systems (NeurIPS)*, pages 7793–7804, 2020.
 - [53] Sunil Kumar Maurya, Xin Liu, and Tsuyoshi Murata. Improving graph neural networks with simple architecture design. *arXiv preprint arXiv:2105.07634*, 2021.
 - [54] Fan Xu, Nan Wang, Hao Wu, Xuezhi Wen, Xibin Zhao, and Hai Wan. Revisiting graph-based fraud detection in sight of heterophily and spectrum. In *Proceedings of the AAAI Conference on Artificial Intelligence (AAAI)*, volume 38, pages 9214–9222, 2024.
 - [55] Yue-Hua Feng and Shao-Wu Zhang. Prediction of drug-drug interaction using an attention-based graph neural network on drug molecular graphs. *Molecules*, 27(9):3004, 2022.
 - [56] Hang Liu, Lijuan Liu, Dahan Wang, and Chaoqun Hong. Multi-hop aware graph convolutional network and collaborative transformer for traffic flow prediction. In *Proceedings of International Conference on Intelligent Computing (ICIC)*, pages 427–438. Springer, 2025.
 - [57] Amit Roy, Kashob Kumar Roy, Amin Ahsan Ali, M Ashraf Amin, and AKM Mahbubur Rahman. Sst-gnn: simplified spatio-temporal traffic forecasting model using graph neural network. In *Proceedings of Pacific-Asia Conference on Knowledge Discovery and Data Mining (PAKDD)*, pages 90–102. Springer, 2021.
 - [58] Keyulu Xu, Chengtao Li, Yonglong Tian, Tomohiro Sonobe, Ken-ichi Kawarabayashi, and Stefanie Jegelka. Representation learning on graphs with jumping knowledge networks. In *Proceedings of International Conference on Machine Learning (ICML)*, pages 5453–5462. PMLR, 2018.
 - [59] Guangtao Wang, Rex Ying, Jing Huang, and Jure Leskovec. Multi-hop attention graph neural network, 2020.
 - [60] Zhengyu Fang, Xiaoge Zhang, Anyin Zhao, Xiao Li, Huiyuan Chen, and Jing Li. Recent developments in gnns for drug discovery. *arXiv preprint arXiv:2506.01302*, 2025.
 - [61] Di Jin, Yingli Gong, Zhiqiang Wang, Zhizhi Yu, Dongxiao He, Yuxiao Huang, and Wenjun Wang. Graph neural network for higher-order dependency networks. In *Proceedings of the World Wide Web Conference (WWW)*, pages 1622–1630. ACM, 2022.
 - [62] Yimeng Min, Frederik Wenkel, and Guy Wolf. Scattering GCN: Overcoming oversmoothness in graph convolutional networks. In *Proceedings of Advances in Neural Information Processing Systems (NeurIPS)*, volume 33, pages 14498–14508, 2020.
 - [63] Jie Zhou, Ganqu Cui, Shengding Hu, Zhengyan Zhang, Cheng Yang, Zhiyuan Liu, Lifeng Wang, Changcheng Li, and Maosong Sun. Graph neural networks: A review of methods and applications. *AI open*, pages 57–81, 2020.
 - [64] Uri Alon and Eran Yahav. On the bottleneck of graph neural networks and its practical implications. *arXiv preprint arXiv:2006.05205*, 2020.
 - [65] Keyulu Xu, Weihua Hu, Jure Leskovec, and Stefanie Jegelka. How powerful are graph neural networks? *arXiv preprint arXiv:1810.00826*, 2018.
 - [66] Trang Pham, Truyen Tran, Dinh Phung, and Svetha Venkatesh. Column networks for collective classification. In *Proceedings of the AAAI Conference on Artificial Intelligence (AAAI)*, volume 31, 2017.
 - [67] Francesco Cambria, Francesco Invernici, Anna Bernasconi, and Stefano Ceri. Data analytics algorithms in property graph databases: A survey. *Computer Science Review*, 62:100981, 2026.
 - [68] Qinghua Wu and Jin-Kao Hao. A review on algorithms for maximum clique problems. *European Journal of Operational Research (EJOR)*, 242(3):693–709, 2015.
 - [69] Maximilien Danisch, Oana Balalau, and Mauro Sozio. Listing k-cliques in sparse real-world graphs. In *Proceedings of the World Wide Web Conference (WWW)*, pages 589–598. ACM, 2018.
 - [70] Fangzhou Ye, Lingxiang Yin, Amir Ghazizadeh Ahsaei, and Hao Zheng. Egma: Enhancing data reuse and workload balancing in message passing gnn acceleration via gram matrix optimization. In *Proceedings of the 61st ACM/IEEE Design Automation Conference (DAC)*, pages 1–6, 2024.
 - [71] Tianfeng Liu, Yangrui Chen, Dan Li, Chuan Wu, Yibo Zhu, Jun He, Yanghua Peng, Hongzheng Chen, Hongzhi Chen, and Chuanxiong Guo. BGL-GPU-Efficient GNN training by optimizing graph data I/O and preprocessing. In *Proceedings of USENIX Symposium on Networked Systems Design and Implementation (NSDI)*, pages 103–118, 2023.
 - [72] Yuke Wang, Boyuan Feng, Zheng Wang, Tong Geng, Kevin Barker, Ang Li, and Yufei Ding. MGG: Accelerating graph neural networks with Fine-Grained Intra-Kernel Communication-Computation pipelining on Multi-GPU platforms. In *Proceedings of USENIX Symposium on Operating Systems Design and Implementation (OSDI)*, pages 779–795, 2023.
 - [73] Yoongu Kim, Weikun Yang, and Onur Mutlu. Ramulator: A fast and extensible dram simulator. In *IEEE Computer Architecture Letters (CAL)*, pages 45–49. IEEE, 2016.
 - [74] Hongshin Jun, Jinhee Cho, Kangseol Lee, Ho-Young Son, Kwiwook Kim, Hanho Jin, and Keith Kim. Hbm dram technology and architecture. In *Proceedings of IEEE International Memory Workshop (IMW)*, pages 1–4. IEEE, 2017.
 - [75] Naveen Muralimanohar, Rajeev Balasubramanian, and Norman P Jouppi. CACTI 6.0: A tool to understand large caches. *University of Utah and Hewlett Packard Laboratories, Technical Report*, 2009.
 - [76] Jack Choquette, Wishwesh Gandhi, Olivier Giroux, Nick Stam, and Ronny Krashinsky. Nvidia a100 tensor core gpu: Performance and innovation. *IEEE Micro*, 41(2):29–35, 2021.
 - [77] Bingyi Zhang, Rajgopal Kannan, and Viktor Prasanna. BoostGCN: A framework for optimizing GCN inference on FPGA. In *Proceedings of IEEE International Symposium on Field-Programmable Custom Computing Machines (FCCM)*, pages 29–39. IEEE, 2021.
 - [78] Zhilin Yang, William Cohen, and Ruslan Salakhudinov. Revisiting semi-supervised learning with graph embeddings. In *Proceedings of International Conference on Machine Learning (ICML)*, pages 40–48. PMLR, 2016.
 - [79] Benedek Rozemberczki, Carl Allen, and Rik Sarkar. Multi-scale attributed node embedding. *Journal of Complex Networks*, page cnab014, 2021.
 - [80] Hongbin Pei, Bingzhe Wei, Kevin Chen-Chuan Chang, Yu Lei, and Bo Yang. Geom-gcn: Geometric graph convolutional networks. *arXiv preprint arXiv:2002.05287*, 2020.
 - [81] Aleksandar Bojchevski and Stephan Günnemann. Deep gaussian embedding of graphs: Unsupervised inductive learning via ranking. *arXiv preprint arXiv:1707.03815*, 2017.
 - [82] Jure Leskovec and Andrej Krevl. SNAP Datasets: Stanford large network dataset collection. <http://snap.stanford.edu/data>, 2014.
 - [83] George Karypis and Vipin Kumar. Metis: A software package for partitioning unstructured graphs, partitioning meshes, and computing fill-reducing orderings of sparse matrices. 1997.
 - [84] Chen Sun, Chia-Hsin Owen Chen, George Kurian, Lan Wei, Jason Miller, Anant Agarwal, Li-Shuan Peh, and Vladimir Stojanovic. Dsent-a tool connecting emerging photonics with electronics for opto-electronic networks-on-chip modeling. In *Proceedings of 2012 IEEE/ACM Sixth International Symposium on Networks-on-Chip (NOCS)*, pages 201–210. IEEE, 2012.