# A High-performance and Flexible Accelerator for Dynamic Graph Convolutional Networks

Yingnan Zhao
*The George Washington University*
Washington, D.C.
yzhao96@gwu.edu

Ke Wang
*University of North Carolina at Charlotte*
Charlotte, NC
ke.wang@charlotte.edu

Ahmed Louri
*The George Washington University*
Washington, D.C.
louri@gwu.edu

***Abstract*—Dynamic Graph Convolutional Networks (DGCNs) have been applied to various dynamic graph-related applications, such as social networks, to achieve high inference accuracy. Typically, each DGCN layer consists of two distinct modules: a Graph Convolutional Network (GCN) module that captures spatial information, and a Recurrent Neural Network (RNN) module that extracts temporal information from input dynamic graphs. The different functionalities of these modules pose significant challenges for hardware platforms, particularly in achieving high-performance and energy-efficient inference processing. To this end, this paper introduces HiFlex, a high-performance and flexible accelerator designed for DGCN inference. At the architecture level, HiFlex implements multiple homogeneous processing elements (PEs) to perform main computations for GCN and RNN modules, along with a versatile interconnection fabric to optimize data communication and enhance on-chip data reuse efficiency. The flexible interconnection fabric can be dynamically configured to provide various on-chip topologies, supporting point-to-point and multicast communication patterns needed for GCN and RNN processing. At the algorithm level, HiFlex introduces a dynamic control policy that partitions, allocates, and configures hardware resources for distinct modules based on their computational requirements. Evaluation results using real-world dynamic graphs demonstrate that HiFlex achieves, on average, a 38% reduction in execution time and a 42% decrease in energy consumption for DGCN inference, compared to state-of-the-art approaches such as ES-DGCN, ReaDy, and RACE.**

## I. Introduction

Dynamic Graph Convolutional Networks (DGCNs) have been proposed to extend machine learning techniques to dynamic graph-related applications [1]–[6], such as social networks [7], traffic forecasting [8], and many others [9]–[11]. Each DGCN layer consists of a Graph Convolutional Network (GCN) module [12]–[14] followed by a Recurrent Neural Network (RNN) module [4], [6], [8]. The GCN module captures spatial information, while the RNN module extracts temporal information from the input dynamic graphs. The DGCN model, with its two distinct modules, presents significant challenges for hardware platforms in achieving high-performance and energy-efficient inference processing.

A significant amount of previous work has been proposed at both the algorithm and architecture levels [1]–[3], [6], [15] to reduce computation latency and accelerate DGCN inference. At the algorithm level, ES-DGCN [3] partitions and distributes DGCN models based on the applied hardware platform, such as CPUs and GPUs, to minimize memory usage and transfer time. CommonGraph [2] is designed to reduce energy consumption by avoiding the need to process energy-expensive deletions of edges and vertices in dynamic graphs. Instead, CommonGraph converts these deletion operations into additions, as addition operations typically consume less energy than deletion operations. At the architecture level, MEGA [1] is proposed to support the existing CommonGraph framework (ES-DGCN) and enable the concurrent execution of multiple snapshots through a tailored processing workflow. RACE [15] and Ready [6] focus on identifying and eliminating redundant computations within each snapshot and introduce a unified architecture for the sequential execution of GCN and RNN modules within each DGCN layer.

While previous works have made significant achievements in improving performance, current approaches still face several challenges, motivating us to propose a flexible DGCN accelerator. First, existing approaches, such as MEGA, RACE, and ReaDy, mainly focus on enhancing computational efficiency through customized computing engines. However, these designs usually integrate a rigid on-chip interconnection, limiting the potential for optimizing data reuse efficiency. In particular, enhancing the on-chip data reuse of different data types, such as input data, intermediate results, and weight matrices, can substantially reduce off-chip memory access, which remains a critical bottleneck in terms of both performance and energy consumption. Second, current designs execute the two distinct modules sequentially. Given that these modules have different computational requirements, these approaches experience significant underutilization of system resources, further degrading overall performance and energy efficiency.

To this end, this paper proposes HiFlex, a high-performance and flexible hardware accelerator for DGCN inference. HiFlex addresses the two key challenges mentioned earlier through two innovative designs: a versatile interconnection fabric and a dynamic control policy, which are detailed as follows:

- **A reconfigurable accelerator with a versatile interconnection fabric:** HiFlex consists of a versatile interconnection fabric that dynamically connects multiple processing elements (PEs) to fulfill the data communication and computation requirements of each DGCN layer. The interconnection fabric supports various on-chip topologies, such as mesh, ring, and tree, to accommodate diverse data communication patterns, including point-to-point and multicast. This ensures fast and energy-efficient data transmission among PEs.

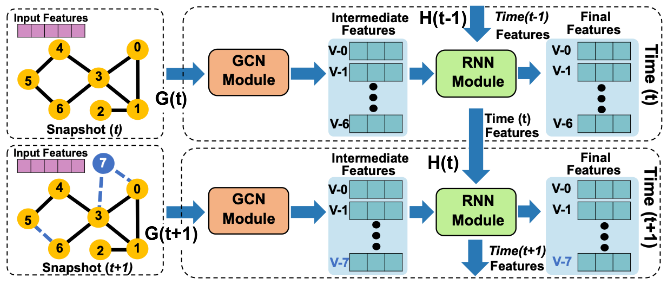

Fig. 1: A typical dynamic graph convolutional network (DGCN) layer includes a graph convolutional network (GCN) module and a recurrent neural network (RNN) module.

- **A dynamic control policy for hardware partitioning and workload allocation:** HiFlex introduces an innovative dynamic control policy to optimize hardware resource utilization, thereby improving performance and energy efficiency. Given an input dynamic graph with a DGCN model, HiFlex first determines the optimal tile size that minimizes on-chip computation and off-chip memory access. Based on this tile size, HiFlex then estimates the computational demands of both GCN and RNN modules, adaptively allocating PEs to each module to maximize hardware resource efficiency.

Evaluation results with real-world dynamic graphs demonstrate that the proposed HiFlex achieves, on average, a 38% execution time saving and 42% energy reduction for DGCN inference as compared to state-of-the-art approaches.

## II. Background and Motivation

### A. DGCN Background

Typically, a DGCN model comprises multiple layers, each consisting of two distinct modules: the graph convolutional network (GCN) module and the recurrent neural network (RNN) module. The GCN module is used to capture structural information, while the RNN module is tasked with capturing temporal information from the input dynamic graphs, as shown in Fig. 1. The main computations of each DGCN layer can be abstracted as shown in Eq. 1. Here, $G_t$, $X_t$, and $H_t$ represent the input dynamic graph, the feature matrix, and the final result of the DGCN model for the current timestamp $t$, respectively.

$$\begin{aligned} \mathrm{X}_t &= GCN(\mathrm{G}_t) \\ \mathrm{H}_t &= RNN(\mathrm{H}_{t-1},\, X_t) \end{aligned} \tag{1}$$

**The GCN module:** Each GCN module includes two key computational phases: aggregation and combination phases. During the aggregation phase, each vertex collects feature vectors from its neighbors. In the combination phase, each vertex uses a pre-trained weight matrix to update the local aggregated feature vector. The main computation can be abstracted as a two-phase matrix-matrix multiplication [16], [17], as shown in Eq.2. The $X$, $A$, and $W_{Comb.}$ represent feature, adjacency, and weight matrices, respectively. The ($\sigma$) denotes activation functions, such as Rectified Linear Unit (ReLU) [18].

$$X_{(t+1)} = \sigma(A_{(t)} \cdot X_{(t)} \cdot W_{Comb.}) \tag{2}$$

**The RNN module:** Each RNN module obtains the result of the previous GCN module ($X_t$) and the result from the last snapshot ($H_{t-1}$) as inputs to calculate the output state of the current snapshot $t$. The main computations include matrix-matrix multiplication, element-wise product, and element-wise addition. To illustrate the computations of an RNN module, we use the Gated Recurrent Unit (GRU) as an example, as shown in Eq.3 [8], [19], [20].

$$\begin{aligned} z_t &= \sigma(W_z X_t + U_z H_{t-1} + b_z) \\ r_t &= \sigma(W_r X_t + U_r H_{t-1} + b_r) \\ \widetilde{h}_t &= tanh(W_h X_t + U_h(r_t \odot H_{t-1}) + b_h) \\ h_t &= (1 - z_t) \odot h_{t-1} + z_t \odot \widetilde{h}_t \end{aligned} \tag{3}$$

Specifically, GRU includes three input matrix multiplications by multiplying the representation vector $X_t$ with three input weight matrices $W_{z,r,h}$. GRU uses three hidden matrix multiplications by multiplying hidden vector $H_{t-1}$ with three hidden weight matrices $U_{z,r,h}$. The $z_t$, $r_t$, $\widetilde{h}_t$, and $h_t$ denote the update gate vector, reset gate vector, candidate activation vector, and final output, respectively. The (+) and ($\odot$) denote the additions and products in elemental terms.

### B. Motivation

Several dedicated approaches have been introduced from both the algorithmic and architectural aspects to efficiently accelerate DGCN inference [1]–[3], [6], [15]. However, current approaches still face several key limitations that impede both performance and energy efficiency. Specifically, during each DGCN layer, multiple types of data can be reused through flexible interconnections to enhance on-chip data reuse efficiency and reduce off-chip memory access. For example, as discussed in Sec.II-A, most gate functions in the RNN module share the same input (the output from the previous GCN module) as one of their inputs. In this scenario, the GCN output can be multicast to the processing units handling the RNN module, significantly enhancing data reuse efficiency. Additionally, the GCN and RNN modules experience varying levels of hardware utilization in current approaches. To illustrate this issue, we evaluate previous works using four input graphs (DBLP, Mobile, Academic, WikiData) and report the average utilization of on-chip computational resources for each module in Table I. The underutilization of hardware resources leads to degraded performance and energy efficiency in existing designs.

TABLE I: On-chip Computational Resources Utilization

| Input Graph | GCN module | RNN module |
|---|---|---|
| DBLP | 67% | 44% |
| Mobile | 29% | 48% |
| Academic | 27% | 41% |
| WikiData | 12% | 46% |

To this end, this paper proposes HiFlex, a customized accelerator designed to effectively address the aforementioned limitations and deliver high-performance, energy-efficient DGCN inference. HiFlex introduces two key innovations: a versatile interconnection fabric capable of reusing multiple types of data across PEs, and a dynamic control policy that determines the optimal tile size while adaptively partitioning and allocating on-chip computational resources.

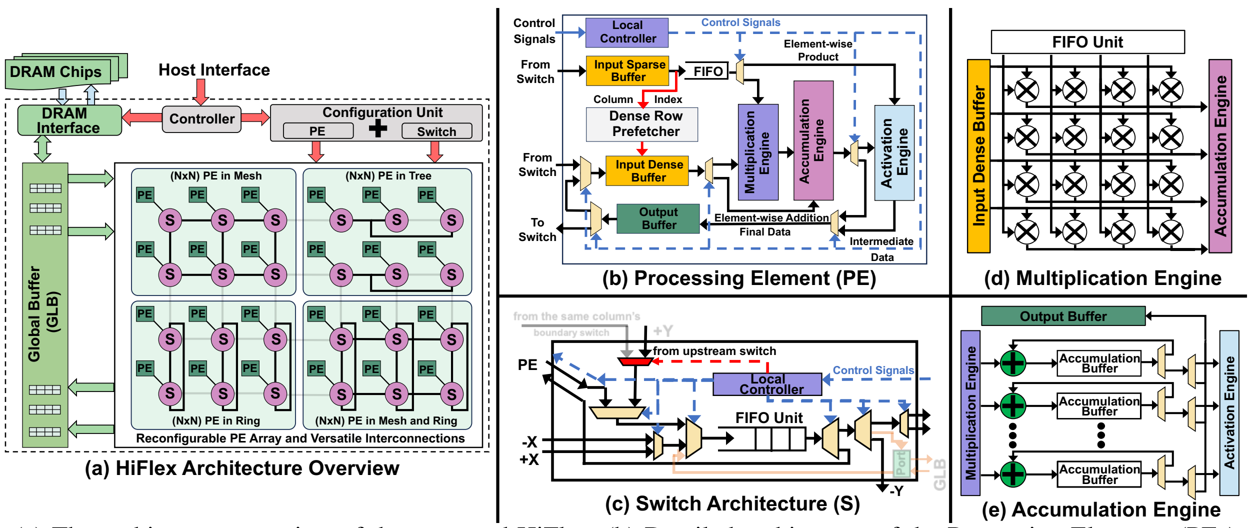

Fig. 2: (a) The architecture overview of the proposed HiFlex. (b) Detailed architecture of the Processing Elements (PEs) that are used to perform the main computations involved in GCN and RNN modules, including Sparse-dense and general matrix-matrix multiplication, as well as element-wise product and addition. (c) Detailed architecture of the switches that are used to support various on-chip topologies, including mesh, ring, and tree. (d) and (e) show details of the multiplication and accumulation engines involved in each PE.

## III. The HiFlex Accelerator

### A. Architecture Overview

As shown in Fig. 2 (a), HiFlex consists of a unified PE array, a versatile interconnection fabric (switches), a global buffer (GLB), a central controller, and a configuration unit. Specifically, HiFlex implements multiple PEs, each can be configured to perform the key computations involved in GCN and RNN modules, including sparse-dense matrix-matrix multiplication (SpMM), general matrix-matrix multiplication (GeMM), element-wise product, and addition. Fig. 2 (b) shows the architecture details of PEs. To manage the data communication among PEs and enhance on-chip data reuse efficiency, HiFlex integrates a versatile interconnection fabric with adaptable switches. Based on the input graph and the DGCN model applied, switches can be dynamically configured to support various types of on-chip topologies, such as mesh, tree, and ring, as detailed in Sec. III-B. The global buffer (GLB) is used to store input, intermediate, and final matrices that are used for computations during DGCN inference processing. The central controller is used to perform the proposed dynamic control policy. Particularly, for a given dynamic graph and DGCN model, the controller first determines the optimal tile size to minimize both on-chip data computations and off-chip data memory access. It then estimates the computational demands of the two distinct modules based on the selected tile size, balancing hardware resources between modules to enhance hardware utilization. Details of the proposed dynamic control policy are introduced in Sec. III-C. All aforementioned configurations are performed per DGCN layers. Once the configuration is completed, data is transferred from the GLB to the corresponding PEs for computations. The detailed time and energy overhead of the configuration setup are provided in Sec. IV-C and Sec. IV-D, respectively.

### B. The Versatile Interconnection Fabric

To manage data communication among PEs and improve on-chip data reuse efficiency, HiFlex implements a versatile interconnection fabric with reconfigurable switches. Fig. 2 (c) provides a detailed overview of the proposed switches. Each switch includes a local controller and a First-In-First-Out (FIFO) unit. The local controller configures MUX-DeMUXes based on the received control signals from the central controller, thereby establishing connections with neighboring switches. The FIFO unit is used to store the received data from either adjacency switches or the connected PE. Based on the basic switch, HiFlex introduces an enhanced switch. The enhanced switch includes a global buffer (GLB) port, as illustrated in Fig. 2 (c), enabling direct retrieval of required data from the GLB. Since the reconfigurable PE array in HiFlex can be scaled to an N×N size, the primary objective of the enhanced switch is to minimize the latency caused by long-distance data transmission. HiFlex organizes its N×N PE array into multiple sub-arrays, with each sub-array consisting of an 8×8 PE array. In this configuration, the boundary switches of each sub-array are enhanced switches, while the remaining internal switches are basic switches.

By dynamically configuring these switches, HiFlex supports a wide range of on-chip topologies, including mesh, tree, and ring. Such adaptability effectively fulfills the distinct data communication requirements induced by both GCN and RNN modules, significantly enhancing on-chip data reuse efficiency and greatly reducing off-chip memory access. As shown in Fig. 3, switches are initially interconnected in a mesh topology.

- **For GCN modules:** As shown in Fig. 3 (a), during the aggregation phase, all switches are interconnected in a mesh topology to manage efficient data communication. To prevent data conflicts during computations, PEs within the same row are allocated to process the same set of vertices. During the combination phase, switches are

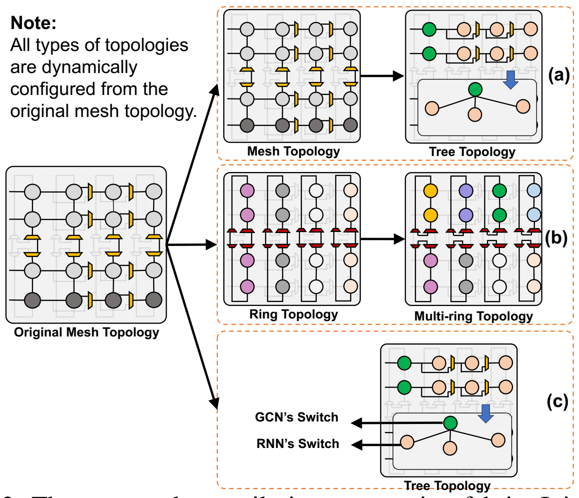

Fig. 3: The proposed versatile interconnection fabric. Initially, all switches (represented as circular components) are interconnected in a mesh topology. (a) Mesh and Tree topologies are used in GCN modules; (b) The ring topology is used in RNN modules. Since the proposed design is scalable, when HiFlex consists of multiple sub-arrays (each with 8x8 PEs), the switches from each sub-array can be combined to form a larger ring topology. (c) The tree topology is used to manage data movement between PEs from GCN and RNN modules.

interconnected in a tree topology as all input vertices share the same small weight matrix in the GCN module [16]. In this configuration, the weight matrix is multicast from the GLB to the corresponding PEs for further calculations.

- **For RNN modules:** As shown in Fig. 3 (b), during the RNN module, all switches are interconnected in a ring topology. PEs within the same column are allocated to perform the specific gate function (as introduced in Sec. II-A) involved in each RNN module. Typically, the weight matrix of the RNN module is significantly larger than that of the GCN module. As a result, the weight matrix is evenly distributed and stored across the PEs in the same column, with PEs using corresponding switches to manage the transmission of weight matrices. To prevent data conflicts and deadlocks during transmission, the vertical ring topology is configured to be directional.
- **Between GCN and RNN modules:** As illustrated in Fig. 3 (c), the switches assigned to the GCN and RNN modules are interconnected in a tree topology. Specifically, the switch assigned to the GCN module functions as the root node, while the switches allocated to the RNN module serve as the leaf nodes. This configuration allows the output of the GCN module from the root PE to be multicast to the PEs assigned to the RNN module within the same row. This is essential because the different gate functions of the RNN modules share the GCN output as input data for their computations, as discussed in Sec. II-A. Given that the latency in the tree topology can increase due to long-distance communication when a large number of PEs are in the same row, HiFlex organizes its N×N PE array into multiple sub-arrays, with each sub-array consisting of an 8×8 PE array.

To ensure data synchronization during matrix-matrix multiplication and eliminate per-hop control timing overhead, all switches are interconnected in a systolic manner, wherein all the tiles of vertices are simultaneously sent across the entire interconnection fabric. This configuration ensures that the interconnection fabric can avoid data conflicts during data transmission without requiring an additional routing algorithm, which may incur extra time and control overhead.

## *C. The Dynamic Control Policy*

Given the varying sizes and sparsity of input dynamic graphs, along with the different hardware resource requirements of the GCN and RNN modules, HiFlex uses a central controller with an innovative dynamic control policy to enhance hardware utilization and, consequently, improve overall performance and energy efficiency. The central controller, along with the proposed control policy, monitors on-chip performance metrics and dynamic partitions and allocates hardware resources to both modules. The primary workflow can be summarized in the following two main steps:

**Step-1 Finding the tile size:** The tile size $<T_N,\ T_K,\ T_C>$ determines the amount of data that can be processed by the PEs in parallel during each epoch. Here, $T_N$ represents the number of vertices processed, $T_K$ denotes the length of the feature vector, and $<T_K \times T_C>$ refers to the size of the weight matrix, either from the GCN or RNN module. Given an input graph with *N* vertices and *K* features, HiFlex estimates the total number of off-chip memory access (DRAM) for each legal tile size in both the GCN and RNN modules. For instance, given a feature matrix (X$\in$ $R^{(N\times K)}$) and a weight matrix (W$\in$ $R^{(K\times C)}$), HiFlex estimates the DRAM access needed for the combination phase followed by Eq.4. As the design space of all legal tile sizes is polynomial, HiFlex offers a set of options based on the number of multipliers and the size of buffers in each PE to streamline the design space. Additionally, HiFlex uses a binary search algorithm with a time complexity of *O(log(n))* to minimize the search time for the optimal tile size within the given set.

$$\begin{aligned}
&DRAM_{total} = DRAM_{feature} + DRAM_{weight} + DRAM_{output}\\
&DRAM_{feature} =(\lceil \tfrac{N}{T_N} \rceil \times \lceil \tfrac{K}{T_K} \rceil \times \lceil \tfrac{C}{T_C} \rceil) \times (T_N \times T_K)\\
&DRAM_{weight} = (\lceil \tfrac{N}{T_N} \rceil \times \lceil \tfrac{K}{T_K} \rceil \times \lceil \tfrac{C}{T_C} \rceil) \times (T_K \times T_C)\\
&DRAM_{output} = (\lceil \tfrac{N}{T_N} \rceil \times \lceil \tfrac{C}{T_C} \rceil) \times (T_N \times T_C)
\end{aligned} \tag{4}$$

**Step-2 Finding the partitioning ratio:** After determining the tile size, HiFlex estimates the number of computations required by the GCN and RNN modules followed by Eq.5 to determine the partitioning ratio (r). Based on the ratio r, PEs are divided into two sections and allocated to the GCN and RNN modules according to the calculated ratio. The corresponding control signals are then selected and forwarded from the global configuration unit to the relevant PEs and switches to configure the system accordingly.

$$\begin{aligned}
&PartitioningRatio(r) = N(GCN) : N(RNN)\\
&\qquad N(GCN) = \textstyle\prod(T_n, T_{k1}, T_n) + \prod(T_n, T_{c1}, T_{k1})\\
&\qquad N(RNN) = \textstyle\prod(T_n, T_{k2}, T_n) + \prod(T_n, T_{c2}, T_{k2})
\end{aligned} \tag{5}$$

TABLE II: Characteristic Statistics of the Input Dynamic Graphs

| Dynamic Graphs | #Vertices | #Edges | #Features | Change ratio of V | Change ratio of E | Descriptions |
|---|---|---|---|---|---|---|
| Mobile | 340,751 | 2,200,203 | 13,452 | 0.7%-1.4% | 1.1%-2.1% | Social Graph |
| DBLP | 315,159 | 1,615,400 | 25,468 | 0.8%-1.3% | 0.9%-1.9% | Academic Graph |
| Academic | 51,060 | 794,551 | 2,849 | 1.2%-2.1% | 0.6%-1.4% | Academic Graph |
| Wikidata | 11,134 | 150,779 | 1,572 | 1.5%-2.9% | 0.2%-2.1% | Knowledge Graph |

Compared to existing approaches from both the algorithm and architecture levels, HiFlex seeks to balance memory access latency, hardware utilization, and on-chip computations, thereby enhancing overall performance and energy efficiency.

## IV. Evaluation and Analysis

### *A. Evaluation Setup*

**Hardware simulator.** To evaluate the performance of the proposed accelerator, we built a cycle-accurate simulator in C++ language. The simulator counts the number of the on-chip arithmetic operations performed by each PE and the on-chip data memory accesses. The number of arithmetic operations is used to calculate the computation time while the number of on-chip memory accesses is used to compute the on-chip communication time. HiFlex uses DRAMsim2 [21] to estimate the time of off-chip data communication. The overall execution time is derived by adding up the off-chip communication time, the on-chip communication time, and the computation time, considering the overlap caused by buffering from different memory hierarchies.

Additionally, the simulator uses the number of on/off-chip data memory access and on-chip computations to estimate the related energy consumption according to [16], [22]. To measure the area and power consumption, we model all key components, including the PEs, switches, global controller, and configuration unit. We use the Synopsys Design Compiler with the TSMC 45nm library for synthesis and estimate power consumption using Synopsys PrimeTime PX. The energy consumption of the control logic is calculated by multiplying their power by execution time (in cycles). We set the clock frequency at 1 GHz. We use Cacti 6.0 [23] to estimate the area, and power consumption, as well as the access latency of all types of on-chip buffers and FIFOs.

**Architecture Configuration.** The proposed HiFlex implements an 8x8 PE array interconnected through a versatile fabric composed of 8x8 switches. As introduced in Sec. III-B, the PE array can be scaled up to an NxN array by combining multiple 8x8 PE arrays. Additionally, each PE consists of a 4x4 multiplier array connected to an accumulation buffer with 8 (2x4) adders. The input sparse and dense buffers within PEs are 320KB and 4KB, respectively. The output buffer is 256KB. The capacity of the FIFO within both types of switches is 320KB, designed to accommodate the largest tile of data that needs to be transferred between PEs such as feature matrices. The size of the global buffer is 2MB to store input, intermediate, and output data during processing.

**Datasets and Benchmarks.** Table II illustrates the detailed information of four dynamic graphs used for evaluation [4], [15], [24], [25]. Given that the updated statistics for both vertices and edges vary, we conducted 100 evaluations for each

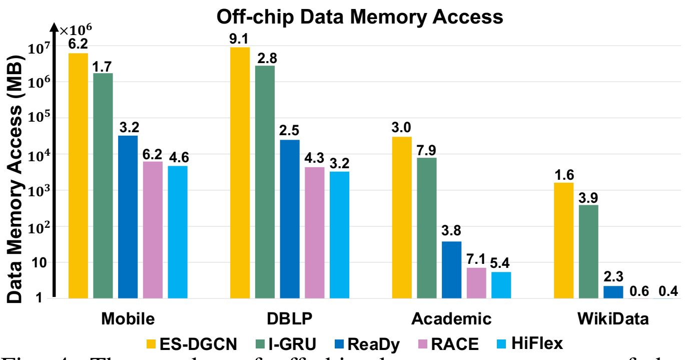

Fig. 4: The number of off-chip data memory access of the proposed HiFlex compared to previous approaches across four input dynamic graphs.

dataset by randomly selecting ratios from the ranges indicated in the table. The average of these results is taken as the final outcome. We select one of the most popular models, named T-GCN [8], which includes a GCN module followed by a GRU module in each layer. It is important to note that HiFlex is also compatible with other models, such as CD-DGCN [26], as they similarly incorporate GCN and RNN modules in each layer.

**Baselines.** We compare the proposed HiFlex to the existing approaches from both algorithm and architecture aspects with the same simulator. Specifically, from the algorithm level, we compare HiFlex to ES-DGCN [3] with a static partitioning algorithm on a CPU/GPU platform. From the architecture level, we use the current customized DGCN accelerators, named ReaDy [6] and RACE [15] as baselines. Additionally, since each DGCN layer includes two distinct modules, named GCN and RNN, we compare the proposed HiFlex to the combination of state-of-the-art GCN (I-GCN [27]) and RNN (DeltaRNN [28]) accelerators, connecting as a tandem-engine architecture (denoted as I-GRU). The main memory bandwidth for all accelerators is scaled to 256GB/s for fair comparison.

### *B. Off-chip Data Memory Access*

Fig. 4 shows the off-chip data memory access of the proposed HiFlex compared to existing approaches across four input dynamic graphs. In contrast to approaches like ES-DGCN, which optimize at the algorithm level and use general hardware platforms such as CPUs, HiFlex integrates PEs capable of directly processing sparse input, thereby minimizing data memory access. Compared to the existing customized DGCN accelerators, HiFlex integrates a versatile interconnection fabric. Specifically, the fabric operates on a mesh topology during the GCN module to enhance input data reuse, while it switches to a ring topology during the RNN module to improve the reuse of weight matrices. Additionally, the interconnection fabric adopts a tree topology to optimize the reuse of intermediate data between the two modules. With enhanced data reuse

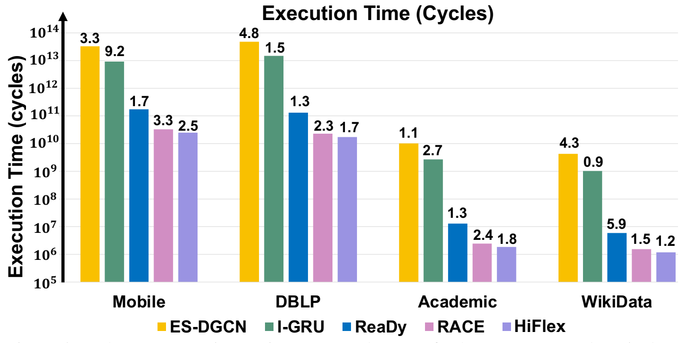

Fig. 5: The execution time (cycles) of the proposed HiFlex compared to previous approaches on four input real-world dynamic graphs.

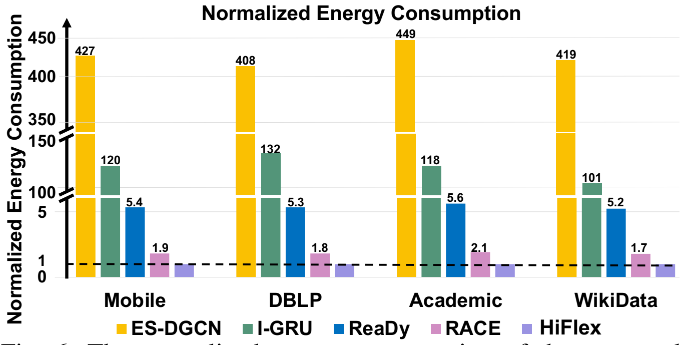

Fig. 6: The normalized energy consumption of the proposed HiFlex compared to previous approaches on four input real-world dynamic graphs.

efficiency, HiFlex significantly reduces the need to access the off-chip memory frequently, as PEs can retrieve data from their neighbors. Fig.4 shows that HiFlex achieves an average reduction of 31% in data memory access compared to RACE.

## C. *Execution Time*

Fig. 5 illustrates the total execution time of the proposed HiFlex compared to previous works, measured by the total number of cycles. The execution time of HiFlex includes the overhead incurred by the controller when performing the control policy and configuring both the switches and the PEs. Due to significantly reduced off-chip memory access, a major performance bottleneck, HiFlex achieves a 38% reduction in execution time (1.6x speedup) on average across four input dynamic graphs, compared to the latest customized accelerator, RACE, which currently offers the best performance among existing approaches.

In terms of time overhead, the controller performs the proposed control policy for each DGCN layer. Specifically, given an input graph, the control policy searches for the optimal tile size from 8 available choices, taking 27 cycles (3 cycles per candidate plus 3 additional cycles for searching and comparison). Additionally, the controller requires 9 cycles to configure all switches and PEs. Thus, the total overhead is approximately 36 cycles. However, since the control policy is applied at the DGCN layer level, this overhead is negligible compared to the execution time of the entire layer and can also overlap. Specifically, while the controller is computing, the features of the input vertices are concurrently loaded from off-chip memory into the global buffer.

## D. *Energy Consumption*

Fig. 6 shows the normalized energy consumption of the proposed HiFlex compared to previous works across four evaluated input graphs. HiFlex achieves an average 42% reduction in energy consumption compared to the current state-of-the-art approach, RACE, across all input dynamic graphs. This energy reduction is primarily due to two factors. First, HiFlex significantly improves the on-chip data reuse efficiency through its versatile interconnection with reconfigurable switches, while also minimizing off-chip memory access. Second, by dynamically partitioning and allocating on-chip hardware resources according to the computational demands of different modules, HiFlex enhances hardware utilization and reduces overall execution time, further lowering energy consumption.

Regarding energy consumption overhead, the controller operates only when executing the control policy and selecting the appropriate control signals for the PEs and switches. Additionally, the configuration unit is accessed solely when configuring the PE array and the interconnection fabric (switches). These operations together introduce a minimal energy overhead of just 3.2% of the total energy consumption.

## E. *Area Analysis*

For area evaluation, we model the proposed accelerator (HiFlex) with the TSMC 45nm library. The evaluation result shows that HiFlex occupies a total area of 198.6($mm^2$). Specifically, the PE array, the interconnection fabric (switches), and the global buffer account for approximately 65.2%, 24.7%, and 5.1% of the total architecture area, respectively. Compared to previous works, HiFlex introduces extra control and data links, as well as MUX-DeMUXes to each PE, to support the various computations required by GCN and RNN modules. Furthermore, HiFlex adds control links to the switches to enable multiple on-chip topologies. Despite these hardware additions, the total area overhead is minimal, inducing only a 0.5% increase in the architecture's area.

## V. Conclusion

We propose HiFlex, a high-performance and flexible accelerator for DGCN inference. HiFlex integrates a versatile interconnection fabric using reconfigurable switches, dynamically connecting PEs to efficiently fulfill the data communication requirements of different modules (GCN and RNN). Additionally, HiFlex introduces a dynamic control policy that optimizes hardware resource utilization by partitioning and assigning on-chip resources to modules based on their computational needs. The proposed HiFlex outperforms previous approaches across four dynamic graphs in terms of off-chip memory access, execution time, and energy consumption.

## Acknowledgment

This research was partially supported by NSF grants CCF-195398, CCF-2131946, CCF-1901165, CNS-2321224, and CNS-2321225.

## References

[1] Chao Gao et al. Mega evolving graph accelerator. In *Proceedings of the 56th Annual IEEE/ACM International Symposium on Microarchitecture*, pages 310–323, 2023.

[2] Mahbod Afarin et al. Commongraph: Graph analytics on evolving data. In *Proceedings of the 28th ACM International Conference on Architectural Support for Programming Languages and Operating Systems, Volume 2*, pages 133–145, 2023.

[3] Venkatesan T Chakaravarthy et al. Efficient scaling of dynamic graph neural networks. In *Proceedings of the International Conference for High Performance Computing, Networking, Storage and Analysis*, pages 1–15, 2021.

[4] Aldo Pareja et al. Evolvegcn: Evolving graph convolutional networks for dynamic graphs. In *Proceedings of the AAAI conference on artificial intelligence*, volume 34, pages 5363–5370, 2020.

[5] Yingnan Zhao et al. An efficient hardware accelerator design for dynamic graph convolutional network (dgcn) inference. In *Proceedings of the 61st ACM/IEEE Design Automation Conference*, pages 1–6, 2024.

[6] Yu Huang et al. Ready: A reram-based processing-in-memory accelerator for dynamic graph convolutional networks. *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, 41(11):3567–3578, 2022.

[7] Avery Ching et al. One trillion edges: Graph processing at facebook-scale. *Proceedings of the VLDB Endowment*, 8(12):1804–1815, 2015.

[8] Ling Zhao et al. T-gcn: A temporal graph convolutional network for traffic prediction. *IEEE transactions on intelligent transportation systems*, 21(9):3848–3858, 2019.

[9] Raymond Cheng et al. Kineograph: taking the pulse of a fast-changing and connected world. In *Proceedings of the 7th ACM european conference on Computer Systems*, pages 85–98, 2012.

[10] Jonathan M Stokes et al. A deep learning approach to antibiotic discovery. *Cell*, 180(4):688–702, 2020.

[11] Keval Vora et al. Kickstarter: Fast and accurate computations on streaming graphs via trimmed approximations. In *Proceedings of the twenty-second international conference on architectural support for programming languages and operating systems*, pages 237–251, 2017.

[12] Justin Gilmer et al. Neural message passing for quantum chemistry. In *International conference on machine learning*, pages 1263–1272. PMLR, 2017.

[13] Thomas N Kipf and Max Welling. Semi-supervised classification with graph convolutional networks. *arXiv preprint arXiv:1609.02907*, 2016.

[14] Yingnan Zhao et al. Opt-gcn: A unified and scalable chiplet-based accelerator for high-performance and energy-efficient gcn computation. *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, 2024.

[15] Hui Yu et al. Race: An efficient redundancy-aware accelerator for dynamic graph neural network. *ACM Transactions on Architecture and Code Optimization*, 20(4):1–26, 2023.

[16] Jiajun Li et al. Gcnax: A flexible and energy-efficient accelerator for graph convolutional neural networks. In *2021 IEEE International Symposium on High-Performance Computer Architecture (HPCA)*, pages 775–788. IEEE, 2021.

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

[18] Bo Jiang et al. Semi-supervised learning with graph learning-convolutional networks. In *Proceedings of the IEEE/CVF conference on computer vision and pattern recognition*, pages 11313–11320, 2019.

[19] Rui Fu et al. Using lstm and gru neural network methods for traffic flow prediction. In *2016 31st Youth academic annual conference of Chinese association of automation (YAC)*, pages 324–328. IEEE, 2016.

[20] JWC Van Lint et al. Freeway travel time prediction with state-space neural networks: Modeling state-space dynamics with recurrent neural networks. *Transportation Research Record*, 1811(1):30–39, 2002.

[21] Paul Rosenfeld et al. Dramsim2: A cycle accurate memory system simulator. *IEEE computer architecture letters*, 10(1):16–19, 2011.

[22] Mark Horowitz. Energy table for 45nm process. In *Stanford VLSI wiki*. 2014.

[23] Naveen Muralimanohar et al. Cacti 6.0: A tool to understand large caches. *University of Utah and Hewlett Packard Laboratories, Tech. Rep*, 147, 2009.

[24] Palash Goyal et al. Dyngem: Deep embedding method for dynamic graphs. *arXiv preprint arXiv:1805.11273*, 2018.

[25] Ryan Rossi and Nesreen Ahmed. The network data repository with interactive graph analytics and visualization. In *Proceedings of the AAAI conference on artificial intelligence*, volume 29, 2015.

[26] Franco Manessi, Alessandro Rozza, and Mario Manzo. Dynamic graph convolutional networks. *Pattern Recognition*, 97:107000, 2020.

[27] Tong Geng et al. I-gcn: A graph convolutional network accelerator with runtime locality enhancement through islandization. In *MICRO-54: 54th annual IEEE/ACM international symposium on microarchitecture*, pages 1051–1063, 2021.

[28] Chang Gao et al. Deltarnn: A power-efficient recurrent neural network accelerator. In *Proceedings of the 2018 ACM/SIGDA International Symposium on Field-Programmable Gate Arrays*, pages 21–30, 2018.