IMA-GNN: In-Memory Acceleration of Centralized and Decentralized Graph Neural Networks at the Edge

Mehrdad Morsali
Department of Electrical and Computer Engineering,
New Jersey Institute of Technology,
Newark, New Jersey, USA
mm2772@njit.edu

Mahmoud Nazzal
Department of Electrical and Computer Engineering,
New Jersey Institute of Technology,
Newark, New Jersey, USA
mn69@njit.edu

Abdallah Khreishah
Department of Electrical and Computer Engineering,
New Jersey Institute of Technology,
Newark, New Jersey, USA
abdallah@njit.edu

Shaahin Angizi
Department of Electrical and Computer Engineering,
New Jersey Institute of Technology,
Newark, New Jersey, USA
shaahin.angizi@njit.edu

ABSTRACT

In this paper, we propose IMA-GNN as an In-Memory Accelerator for centralized and decentralized Graph Neural Network inference, explore its potential in both settings and provide a guideline for the community targeting flexible and efficient edge computation. Leveraging IMA-GNN, we first model the computation and communication latencies of edge devices. We then present practical case studies on GNN-based taxi demand and supply prediction and also adopt four large graph datasets to quantitatively compare and analyze centralized and decentralized settings. Our cross-layer simulation results demonstrate that on average, IMA-GNN in the centralized setting can obtain ~790× communication speed-up compared to the decentralized GNN setting. However, the decentralized setting performs computation ~1400× faster while reducing the power consumption per device. This further underlines the need for a hybrid semi-decentralized GNN approach.

CCS CONCEPTS

• **Computer systems organization → Neural networks**.

KEYWORDS

graph neural network, in-memory computing, edge computing

ACM Reference Format:
Mehrdad Morsali, Mahmoud Nazzal, Abdallah Khreishah, and Shaahin Angizi. 2023. IMA-GNN: In-Memory Acceleration of Centralized and Decentralized Graph Neural Networks at the Edge. In *Proceedings of the Great Lakes Symposium on VLSI 2023 (GLSVLSI '23), June 5–7, 2023, Knoxville, TN, USA.* ACM, New York, NY, USA, 6 pages. https://doi.org/10.1145/3583781.3590248

GLSVLSI '23, June 5–7, 2023, Knoxville, TN, USA

ACM ISBN 979-8-4007-0125-2/23/06…$15.00
https://doi.org/10.1145/3583781.3590248

1 INTRODUCTION

Graph data structures appear naturally in many fields such as user accounts in a social network, atoms in a chemical molecule, and vehicles in a traffic system. Graph Neural Networks (GNN) extend deep learning to graph data by combining graph structure and node information through message passing and aggregation [8, 20]. This enables GNNs to deliver node embeddings to serve multiple downstream graph tasks such as node classification (inferring the class label of a node), link prediction (estimating the possibility of a link between given nodes), and graph classification (inferring the class label of a graph). Essentially, GNNs obtain embeddings for the nodes in a given graph. Each node has a computational graph composed of its k-hop neighboring nodes. Node embeddings are obtained by alternating between message passing, i.e., communicating local information across nodes, and aggregation where received messages along with previous node information are used to obtain an updated embedding. Message passing is done according to the graph structure, whereas aggregation is done by the (trainable) neural network layers of the GNN model [10, 22]. As shown in Fig. 1, for a sample input graph G, first, in the aggregation stage, each node aggregates the information from all neighbors (nodes 2, 4, 7, 8, 9) with its own data (node 3) and creates the Z matrix that represents the aggregated node features from node 3 and its neighbors. During the feature extraction stage, the result of the aggregation stage (Z) is fed into a Multi-Layer Perceptron (MLP) or a Convolutional Neural Network (CNN) model to generate the output matrix indicated by O. The GNN workflow performs the same steps for all the nodes in the graph.

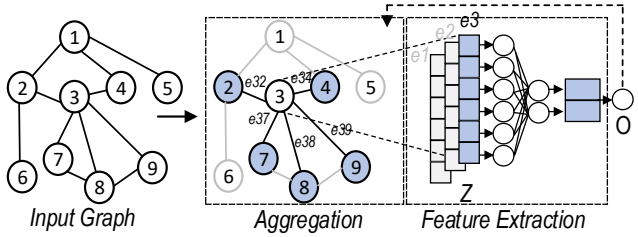


Figure 1: GNN's aggregation and feature extraction stages for a sample input graph.

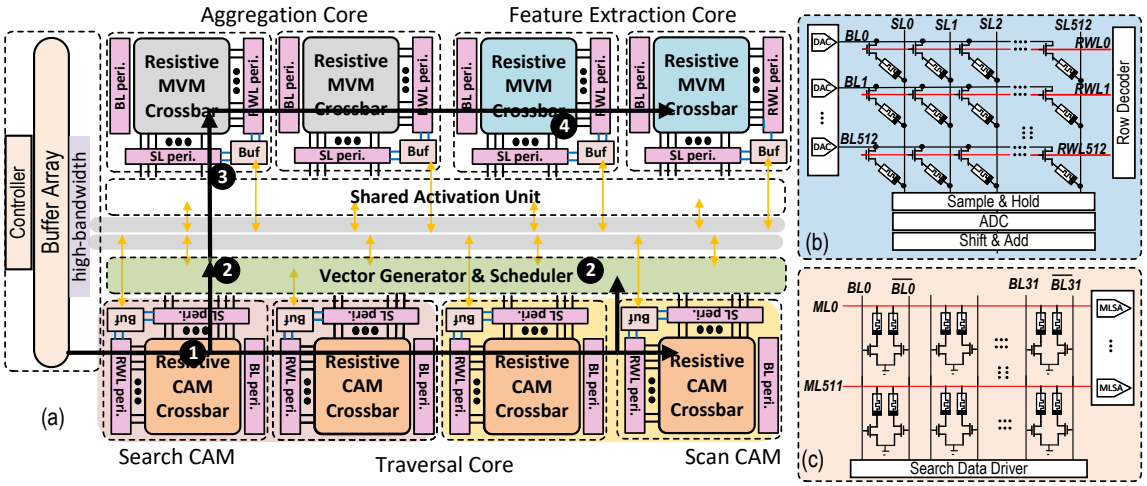


Figure 2: (a) The proposed IMA-GNN architecture with resistive CAM traversal core, resistive MVM aggregation, and feature extraction cores, (b) Resistive MVM crossbar, (c) Resistive CAM crossbar.

Graphs in many real-world application areas are naturally enormous. For example, social media and e-commerce graphs such as Facebook and Amazon graphs have billions of nodes and edges. Furthermore, local node features are typically of large dimensions ranging between hundreds to several thousands [9]. This creates a corresponding time and memory space demand for training and testing on the underlying GNN models [24, 25]. This is because nodes are mutually dependent, and therefore, graphs can not be arbitrarily divided into smaller subgraphs. Techniques such as neighborhood sampling [7] may help to some extent, but depending on the graph structure, even a sampled computation graph and associated features may not fit in the memory of a single GPU. Besides, off-chip memory access is a critical issue in the Von-Neumann computing architecture [2, 3, 11, 16] and to minimize the latency and power consumption, the Processing-in-Memory (PIM) accelerators have been set forth focusing on centralized GNNs. Along this line, HyGNN [22] supports hybrid computing and memory access patterns and performs GCN computations efficiently using a hybrid architecture. Utilizing two dedicated processing engines, HyGNN tackles irregularity with an aggregation engine and leverages regularity with a combination engine. The irregularity arises from the topology-dependent aggregation process which is inherently random and sparse, while the transformation process is a neural network operation that has a static and regular execution pattern. In the same way, GRIP [10] divides the GNN computations into different engines. Multiple parallel prefetch and reduction engines have been used for aggregation to alleviate the irregularity. For regular computation and memory access patterns, GRIP utilizes a high-performance matrix multiply engine with dedicated memory. Another edge-centric paradigm, EnGN [14], has been also implemented using a ring-edge-reduce dataflow. EnGN alleviates the poor locality of sparse and random connected vertices (nodes). AWB-GCN [6] proposes a hardware-based workload distribution auto-tuning framework consisting of three workload rebalancing techniques to alleviate the extreme workload imbalance. PIM-GCN [4] presents a node-stationary dataflow with support for compressed sparse row and column graph representations.

Distributed (decentralized) GNN training and inference [12, 21] is based on dividing a given graph into smaller subgraphs that can be more easily processed with distributed devices. Despite its ability to mitigate the computation overhead by load sharing, decentralized GNN operation faces a major bottleneck; the excessive communication overhead between nodes in different distributed devices [24]. To the best of our knowledge, this work is among the first to explore and compare the in-memory acceleration of centralized and decentralized GNN settings and to offer a design guideline to the community. The contributions of this paper are as follows: (1) We develop a PIM architecture with RRAM arrays based on a set of innovative micro-architectural designs that can be optimized and used for centralized and decentralized GNN inference for efficiency and speed-up; (2) We model the latency and power consumption of GNN accelerators implemented in centralized and decentralized settings considering the computation and communication between edge devices; and (3) We present a bottom-up evaluation framework to analyze the performance of the whole system in real scenarios and through adopting large graph datasets.

2 PROPOSED IMA-GNN

2.1 Architecture Overview

The IMA-GNN is a high-performance and energy-efficient RRAM crossbar-based accelerator developed to execute GNN's pivotal operations in both centralized and decentralized settings inspired by [4]. As shown in Fig. 2(a), IMA-GNN comprises three computation cores, i.e., traversal, aggregation, and feature extraction as well as peripherals such as a buffer array and a controller. The traversal core consists of resistive Content Addressable Memory (CAM) crossbars capable of search and comparison operations (Fig. 2(c)). All resistive CAM crossbars on the bottom side are connected to a shared vector generator & scheduler unit and then to a high-bandwidth bus to communicate with other cores at the top. The aggregation core includes resistive crossbars (Fig. 2(b)) to perform in-situ Matrix-Vector-Multiplication (MVM) operations for the feature aggregation. The feature extraction core is designed with a similar resistive

crossbar but a different size to take care of transformation in GNN inference. The crossbars are connected to a shared activation unit.

2.2 MVM & CAM Crossbars

The RRAM crossbar memory arrays are widely explored as a potential parallel engine to execute MVM operation and scan and search [2–4]. As shown in Fig. 2(b) in the MVM crossbars, the weight parameters are first stored as resistance states in each RRAM device in a 1-Transistor-1-RRAM (1T1R) structure, and then the input binary bit-strings, as the inputs to the crossbar array, are converted by the Digital-to-Analog Converter (DAC) into voltages V_i and applied to Bit-Lines (BLs) in parallel. The weighted currents generated from the RRAM cells sharing a Source-Line (SL) are accumulated resulting in an intrinsic dot-products operation. The accumulated values are then sampled by Sample & Hold unit and then converted to binary data using Analog-to-Digital Converters (ADC). The partial-product results from each SL are further processed by the Shift & Add unit to generate the final result. As shown in Fig. 2(c), in the CAM crossbars, each Ternary CAM (TCAM) cell consists of 2-Transistor-2-RRAM (2T2R) to accomplish the XNOR search operation on each pair of cells. For this operation, BL and $\overline{BL}$s are valued with the search data by the Search Data Driver. Accordingly, the Sense Amplifier connected to Match-Lines (MLSA) senses whether the row is a match or mismatch with the reference connected to Vdd. In the compare operation, BLs are grounded and $\overline{BL}$s are connected to increasing calibrated voltages from the Least Significant Bit (LSB) to the Most Significant Bits (MSB).

2.3 Accelerator Dataflow

Once the edge buffers shown in Fig. 2(a) on the left have been loaded with graph data in either centralized or decentralized GNN settings, the traversal core starts processing edges. The traversal core performs two essential CAM-based operations, i.e., search and compare. To maximize the data reuse of feature data in IMA-GNN, the traversal core implements an efficient node-stationary dataflow by buffering a set of node features in the buffer array and reusing it for the aggregation core. IMA-GNN leverages a Compressed Sparse Row (CSR) format [15] to form the Edge weight array (E), Column Index array (CI), and Row Pointer array (RP) and loads the graph data to search and scan CAMs (Fig. 2(a) ❶). A sample graph adjacency matrix and the corresponding CSR format are shown in Fig. 3(a)-(b). Any destination node then operates as an input to the search CAM as shown in the data mapping in Fig. 3(c) and rows that match the search data are activated. Matching rows are reference inputs for comparison in the scan CAM, which determines the source nodes with edges to the destination node by comparing the input row with RP (Fig. 3(d)). Next, the vector generator & scheduler unit receives the result of scan CAM and edge data to render input control vectors for the aggregation core (Fig. 2(a) ❷). This will activate particular rows of resistive aggregation core corresponding to incoming edges. Next, the aggregation core input buffers receive input vectors for each destination node along with the destination node. The aggregation core starts with source node features or feature dimensions across its own cluster ❸. IMA-GNN is equipped with double buffering for feature data and graph data. This feature enables overlapping writing/programming phases and the traversal stage. Next, the updated destination node features are fed to the feature extraction core's crossbars programmed with weights ❹.

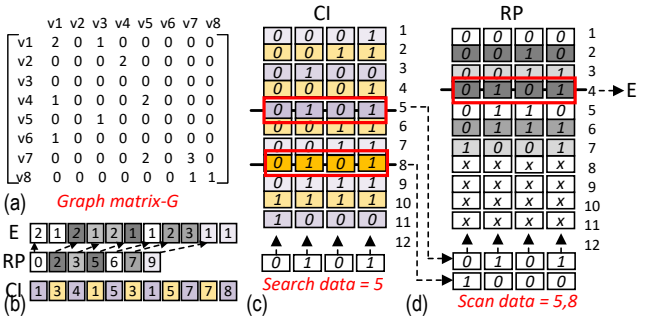


Figure 3: IMA-GNN's hardware mapping and acceleration in traversal core: (a) Sample graph adjacency matrix, (b) CSR, (c) Search CAM operation, (d) Scan CAM operation.

Besides, similar to [16], to maximize the crossbar utilization in the aggregation core, both the aggregation and feature extraction cores could work in parallel.

3 NETWORK MODELING

We explore both centralized and decentralized GNN landscapes to fairly model the performance metrics in both settings. Figure 4 shows a sample graph with N ($N \in \mathbb{Z}^+$) nodes (edge devices). In the centralized GNN setting, a single powerful node as the accelerator is designed with embedded traversal, aggregation, and feature extraction cores to communicate through fast inter-network links (L_n) [17] to aggregate all edge devices' information and handle the computation burdens of transformation. These cores have M_1, M_2, and M_3 times larger allocated computing hardware for traversal, aggregation, and feature extraction operations respectively than a single node in the decentralized mode. Thus, we assume the processing capability of the edge device in the centralized setting is M_1, M_2, and M_3 times larger than the processing capability of a single node in the decentralized mode in the traversal, aggregation, and feature extraction operations, respectively. In the decentralized GNN setting, each edge device is observed as an accelerator with reduced traversal and aggregation cores and in addition to a copy of our network, has an embedded feature extraction core processing L layers. The output of the feature extraction core at each edge device is only communicated to the adjacent edge devices at a defined cluster as shown in Fig. 4(b). Therefore, the communication between neighbors through inter-cluster links (L_c) [18] generates a communication volume as well. The bidirectional communication volume between node-i to node-j is represented as $e_{i,j}$. Therefore, the minimization of the accelerator's computational latency/power and communication latency/power is a pivotal need in the research community. We estimate the centralized and decentralized GNN accelerators' latency as:

$$T_{Net}(N) = T_{compute}(N) + T_{communicate}(N). \quad (1)$$

In an N-edge device graph shown in Fig. 4, denoting by t_1, t_2, and t_3 the traversal, aggregation, and feature extraction cores' latency, respectively, the computation latency of a single node in the decentralized GNNs can be estimated by:

$$T_{compute-decentralized} = t_1 + t_2 + t_3, \quad (2)$$

where in the centralized setting, considering the processing capability of a single powerful edge device the computation latency is

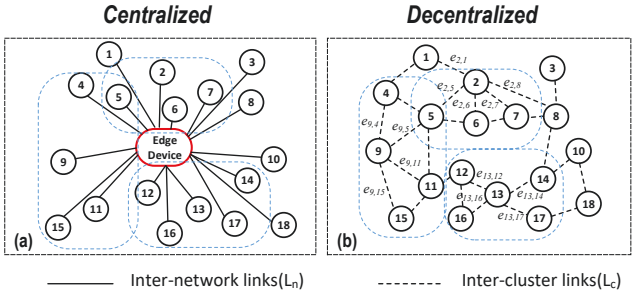


Figure 4: Intra- and inter-edge links in a sample (a) centralized versus (b) decentralized GNN.

given by:

$$T_{compute-centralized} = (t_1/M_1 + t_2/M_2 + t_3/M_3) \times (N-1). \quad (3)$$

In the decentralized setting, the communication latency, $T_{communicate}$, can be given by:

$$T_{communicate-centralized} = (t_e + (c_s \times t(L_c))) \times 2, \quad (4)$$

where t_e is the required time for establishing a connection between two adjacent nodes, c_s denotes the number of adjacent nodes inside a cluster, $t(L_c)$ denotes the latency of the inter-cluster link, and number 2 is to model a two-way link. We assume that data communication inside each cluster is done in a sequential way, thus the number of adjacent nodes is multiplied by the latency of the inter-cluster link. For the centralized setting, we assume data transfer between the central edge device and nodes is done in a concurrent way. Therefore, the communication latency, $T_{communicate}$, for centralized inference can be given by:

$$T_{communicate-decentralized} = t(L_n), \quad (5)$$

where $t(L_n)$ is the latency of the inter-network link. Suppose each edge device runs a GNN with X-layers, the number of input and output activations for a layer x for $1 \leqslant x \leqslant X$ can be given by $\alpha(x)$ and $\alpha(x+1)$, respectively. The total power consumption of GNNs implemented in the proposed accelerator can be developed as:

$$P_{Net}(N) = P_{compute}(N) + P_{communicate}(N). \quad (6)$$

The first part accounts for the computation power and the second part considers communication power for inter-network and inter-cluster links. In the centralized setting, $P_{compute-centralized}$ is given by $\frac{E_{compute-centralized}}{T_{compute-centralized}}$ and $P_{communicate-centralized}$ can be given by $p(L_n) \times 2$. Here $p(L_n)$ denotes the power consumption of the inter-network link and number 2 is to model a two-way transfer. $E_{compute-centralized}$ is readily calculated by achieving the energy consumption values of traversal, aggregation, and feature extraction cores. As for the decentralized setting, $P_{compute-decentralized}$ can be computed with respect to energy and latency parameters. We consider $\sum_{n=1}^{c_s} p_n(c_s(n)(c_s(n)-1))$ transactions between all accelerators inside the cluster. Considering the X-layer GNN, $P_{communicate}$ can be expressed as follow:

$$P_{communicate-decentralized} = \frac{1}{t(L_c)} \times \sum_{x=1}^{X-1} \alpha(x+1) \times E_{perBit} \quad (7)$$

4 EXPERIMENTS

4.1 Evaluation Framework

To evaluate the performance of the proposed architecture, a comprehensive bottom-up evaluation framework is developed as depicted in Fig. 5. At the circuit-level, we use the SPICE model for memristors with the Ag-Si memristor device parameters from [5]. We then combine the SPICE models of CMOS transistors and memristors under NCSU 45nm CMOS PDK [1] to fully design and verify the IMA-GNN cores in HSPICE and to extract performance parameters such as delay and power consumption. We use the Synopsys Design Compiler to develop the controller and buffer array using a standard industry-level 45nm technology. At the architecture-level, we modify and configured the NVSIM-CAM [13] memory evaluation tool and MNSIM [26] with our circuit-level results to extract the performance parameters for traversal, aggregation, and feature extraction cores. The results are then fed to an in-house MATLAB code with the graphs taken as input to calculate the estimated latency and power consumption for various workloads. To have a fair comparison between GNN settings, we set up IMA-GNN's traversal, aggregation, and feature extraction cores with 2K×(512×32), 1K×(512×512), and 256×(128×128), respectively, for the centralized setting and 512×32, 512×512, and 128×128, respectively, for the decentralized setting. It is noteworthy that in addition to the size and number of crossbars in each core, Several factors determine the total latency and power consumption of IMA-GNN in each setting, such as the distribution of graph edges across nodes, the availability of graph data, on-chip storage, and off-chip data accesses.

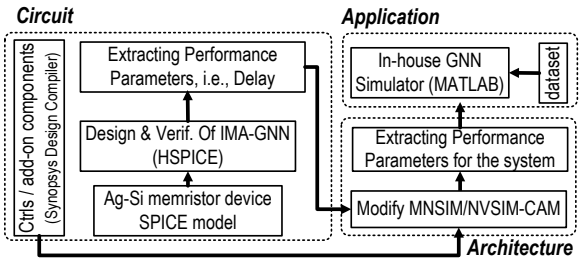
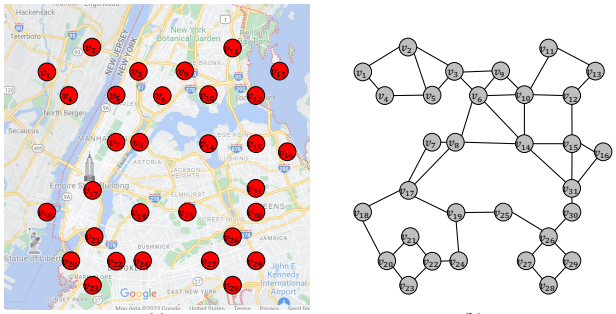


Figure 5: Circuit-to-application evaluation framework.

4.2 Traffic Demand Forecasting

As a case study on the potential of hardware-accelerated decentralized GNNs in real-world applications, we pick a recent work on city-wide multi-relational and spatiotemporal taxi demand and supply forecasting [19]. Figure 6(a) and (b) show sample taxis in a city region and their corresponding graph representation, respectively. This graph is composed of taxi nodes linked by three edge types; *road connectivity*, *location proximity*, and *destination similarity* edges linking taxis connected by a road, being nearby, and targeting nearby destinations, respectively. For each taxi node, the objective is to predict the values of transportation demand and supply for a region surrounding it. This is done based on both historical demands and supplies in the node's surrounding region and the corresponding messages shared by other connected nodes (taxis). Formally, the objective is to train a GNN operator $\mathcal{F}$ that predicts the Q-long future passenger demands and supplies in an m by n region surrounding each taxi at a time instant t, $\mathbf{X}_{t+1:t+Q} \in \mathbf{R}^{Q \times m \times n}$. This is based on the P historical values of the demands and supplies in this region, and the corresponding P historical values passed from the k-hop neighboring nodes. So, $[\mathbf{X}_{t-P+1:t}, G] \xrightarrow{\mathcal{F}} \mathbf{X}_{t+1:t+Q}$, where $\mathbf{X}_{t-P+1:t} \in \mathbf{R}^{P \times m \times n}$, and G denotes the node's computational graph. Due to the existence of multiple edge types and the



(a) (b)

Figure 6: (a) Taxi representation as a graph, and (b) Decentralized GNN operation.

time-dependency, $\mathcal{F}$ is composed of a heterogeneous GNN (hetGNN) used for message passing, followed by a Long-Short-Term Memory (LSTM) network to incorporate time dependency, as depicted in Fig. 7.

The enormous sizes of transportation graphs make it challenging to apply the model in [19] in centralized GNN inference. This limitation is further aggravated by the hetGNNs and the huge volumes of local node information. As a remedy to resolve this limitation, the authors in [19] propose a decentralized GNN inference approach. In this approach, each taxi node has a copy of the model (hetGNN-LSTM), exchanges messages with its k-hop neighbors, and then uses the hetGNN-LSTM model to predict the demands and supplies in its surrounding region. A natural advantage of this approach is handling dynamically varying graph structures. Nevertheless, despite the promising advantages of decentralization, there is still a demanding need for reducing the overall computation and communication latency in the operation of the model. In this experiment, for the centralized GNN setting, the overall latency (in terms of transmission delay) for sending and receiving a packet of 300 Bytes is considered 1.1 ms where the range of the network is 300 meters [17]. This latency is the average overall latency to correctly receive a packet of 300 bytes. Thus, for a packet size of 864 bytes, which is the size of our data, the overall transmission delay can be ~3.3 ms. As for the decentralized GNN setting, we assume that nodes in the graphs of [19] communicate with each other using an ad-hoc wireless network that uses channel 9 (2.452 GHz) of IEEE 802.11n, where the transmission power is fixed to −31 dBm, and bandwidth is 20 MHz. In this configuration, source nodes feed their messages to nearby proxy (relay) nodes which forward the messages to the next nodes, and so on. Since source nodes have more computation compared to proxy (relay) nodes, they incur more

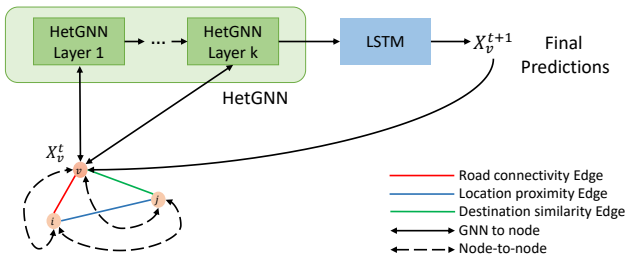


Figure 7: The architecture of the hetGNN-LSTM based prediction in [19].

Table 1: Computation and communication latency/power.

Settings	Centralized		Decentralized	
Figure of merits	Latency	Power	Latency	Power
Traversal	38.43 ns	10.8 mW	7.68 ns	0.21 mW
Aggregation	142.77 μs	780.1 mW	14.27 μs	41.6 mW
Feature extraction	14.53 μs	32.21 mW	0.37 μs	3.68 mW
Computation (Net)	157.34 μs	823.11 mW	14.6 μs	45.49 mW
Communication	3.30 ms	-	406 ms	-

delay. We leverage the proposed bottom-up evaluation framework to estimate IMA-GNN architecture performance in both centralized and decentralized settings. In our evaluation, the number of nodes (taxis) of the graph and the cluster size (c_s) are set as 10000 and 10, respectively. The evaluation results of the latency and power consumption are tabulated in Table 1.

In view of the results, the decentralized setting in [19] improves the total computation latency by a factor of ~10×. This is achieved by reducing the traversal latency, aggregation latency, and feature extraction latency by factors of 5×, 10×, and ~39×, respectively. Thus, a huge improvement can be observed in terms of computation latency. However, in terms of communication, the centralized setting acts much better than the decentralized setting by incurring a ~120× less latency. As for computation power consumption, we observe that the decentralized setting reduces the power budget per node by a factor of 18×. The aggregation core of IMA-GNN consumes most of the power in both centralized and decentralized settings as well as the highest latency. Overall, each of the settings has its own advantage from a different point of view. In the next subsection, more graph datasets with different characteristics are studied in order to further elucidate the pros and cons of centralized and decentralized settings.

4.3 Graph Datasets

In the second case study, four graph datasets, LiveJournal, Collab, Cora, and Citeseer [4, 23] are used to evaluate the inference latency using the proposed IMA-GNN in centralized and decentralized settings. Key graph statistics of these datasets are provided in Table 2. A given vertex is mapped deterministically to a fixed-sized, uniform sample of its neighbors. Figure 8 shows the latency for the four aforementioned graph datasets. Each bar consists of two parts; computation latency and communication latency. For each dataset, we have two bars representing the latency in the centralized (left) and decentralized (right) settings. It can be realized that in all under-test datasets, the computation latency of the decentralized setting is less than that of the centralized setting. Especially, the difference between the computation latency of centralized and decentralized settings is huge in LiveJournal and Collab datasets, where the graph size is much larger than the other two datasets. Given Fig. 8, amongst the four datasets, LiveJournal has the largest computation latency in the centralized settings because it owns the largest number of nodes. This is because in the decentralized

Table 2: Key statistics of the graph datasets used.

Datasets	LiveJournal	Collab	Cora	Citeseer
Number of Nodes	4,847,571	372,475	2708	3,327
Number of Edges	68,993,773	24,574,995	5429	4,732
Feature Length	1	496	1433	3,703
Average C_s	9	263	4	2

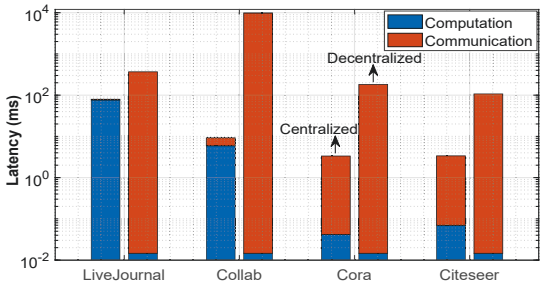


Figure 8: Breakdown of communication and computation latency for centralized and decentralized settings.

mode, as each node is responsible to do its' computation, the computation latency is independent of the total number of nodes and doesn't increase when the number of nodes grows. Conversely, in the centralized setting, by growing the number of nodes, the computation burden of the edge device will increase, causing an increment in the computation latency. On average for these four datasets, the decentralized setting performs computations ~1400× faster. However, the communication latency which is the dominant part of the total latency is much higher in the decentralized setting as each node is required to establish a peer-to-peer connection and transfer data with all adjacent nodes sequentially. According to Fig. 8, Collab has the largest communication latency amongst the four datasets, in the decentralized settings due to its large average C_s, where each node is required to communicate with a large number of adjacent nodes sequentially. However, in the centralized mode, all nodes are connected to the edge device using a fast and mature connection and can transfer data in a parallel way. As for the communication latency, for four under-test datasets, the centralized setting is ~790× faster than the decentralized setting. The performance of the IMA-GNN architecture can increase linearly with an increase in the number of resistive CAM and MVM crossbars in decentralized setting for various datasets and saturate once the entire node feature data could be fitted onto the crossbars. However, it comes at the cost of higher power consumption for each node.

5 CONCLUSION

While the respective benefits of centralized and decentralized GNNs are known in software implementation, there is a lack of hardware implementation analysis to show the communication and computation loads in each setting. This work undertakes this task by modeling and analyzing practical case studies on GNN-based taxi demand and supply prediction and adopting large-scale graph datasets. Our cross-layer simulation results demonstrate our proposed platform called IMA-GNN in the centralized GNN setting can obtain ~790× communication speed-up compared to the decentralized GNN setting. However, the decentralized GNN setting performs computation ~1400× faster while reducing the power consumption per device. This study is conducted based on certain assumptions as discussed. Nevertheless, the results extrapolate that the decentralized GNN setting achieves gains in reducing the computation latency. However, this comes at the expense of increasing communication overhead and latency. This latency is more strongly pronounced with larger graphs. This finding confirms the necessity and the potential of balancing this communication-computation trade-off through a semi-decentralized setting [19].

ACKNOWLEDGMENTS

This work is supported in part by the National Science Foundation under Grant No. 2228028, 2216772, and 1852375.

REFERENCES

[1] 2011. *NCSU EDA FreePDK45.* http://www.eda.ncsu.edu/wiki/FreePDK45
[2] Minhaz Abedin, Arman Roohi, Maximilian Liehr, Nathaniel Cady, and Shaahin Angizi. 2022. MR-PIPA: An Integrated Multilevel RRAM (HfOx)-Based Processing-In-Pixel Accelerator. *IEEE JxCDC* 8, 2 (2022), 59–67.
[3] Shaahin Angizi, Sepehr Tabrizchi, and Arman Roohi. 2022. Pisa: A binary-weight processing-in-sensor accelerator for edge image processing. *arXiv preprint arXiv:2202.09035* (2022).
[4] Nagadastagiri Challapalle, Karthik Swaminathan, Nandhini Chandramoorthy, and Vijaykrishnan Narayanan. 2021. Crossbar based processing in memory accelerator architecture for graph convolutional networks. In *ICCAD*. IEEE, 1–9.
[5] Ligang Gao et al. 2012. Analog-input analog-weight dot-product operation with Ag/a-Si/Pt memristive devices. In *VLSI-SoC*. IEEE, 88–93.
[6] Tong Geng et al. 2020. AWB-GCN: A graph convolutional network accelerator with runtime workload rebalancing. In *MICRO*. IEEE, 922–936.
[7] Will Hamilton, Zhitao Ying, and Jure Leskovec. 2017. Inductive representation learning on large graphs. *Advances in neural information processing systems* 30 (2017).
[8] William L Hamilton, Rex Ying, and Jure Leskovec. 2017. Representation learning on graphs: Methods and applications. *arXiv preprint arXiv:1709.05584* (2017).
[9] Weihua Hu, Matthias Fey, Marinka Zitnik, Yuxiao Dong, Hongyu Ren, Bowen Liu, Michele Catasta, and Jure Leskovec. 2020. Open graph benchmark: Datasets for machine learning on graphs. *Advances in neural information processing systems* 33 (2020), 22118–22133.
[10] Kevin Kiningham, Philip Levis, and Christopher Ré. 2022. GRIP: A graph neural network accelerator architecture. *IEEE Trans. Comput.* (2022).
[11] Jiajun Li, Ahmed Louri, Avinash Karanth, and Razvan Bunescu. 2021. GCNAX: A flexible and energy-efficient accelerator for graph convolutional neural networks. In *HPCA*. IEEE, 775–788.
[12] Qingbiao Li, Fernando Gama, Alejandro Ribeiro, and Amanda Prorok. 2020. Graph neural networks for decentralized multi-robot path planning. In *2020 IROS*. IEEE, 11785–11792.
[13] Shuangchen Li, Liu Liu, Peng Gu, Cong Xu, and Yuan Xie. 2016. Nvsim-cam: a circuit-level simulator for emerging nonvolatile memory based content-addressable memory. In *ICCAD*. ACM, 1–7.
[14] Shengwen Liang, Ying Wang, Cheng Liu, Lei He, LI Huawei, Dawen Xu, and Xiaowei Li. 2020. Engn: A high-throughput and energy-efficient accelerator for large graph neural networks. *IEEE Trans. Comput.* 70, 9 (2020), 1511–1525.
[15] Kai Lu, Zhaoshi Li, Leibo Liu, Jiawei Wang, Shouyi Yin, and Shaojun Wei. 2019. Redesk: A reconfigurable dataflow engine for sparse kernels on heterogeneous platforms. In *ICCAD*. IEEE, 1–8.
[16] Sumit K Mandal, Gokul Krishnan, A Alper Goksoy, Gopikrishnan Ravindran Nair, Yu Cao, and Umit Y Ogras. 2022. COIN: Communication-Aware In-Memory Acceleration for Graph Convolutional Networks. *IEEE JETCAS* 12 (2022), 472–485.
[17] Valerian Mannoni et al. 2019. A comparison of the V2X communication systems: ITS-G5 and C-V2X. In *VTC2019-Spring*. IEEE, 1–5.
[18] Taichi Miya et al. 2021. Experimental analysis of communication relaying delay in low-energy ad-hoc networks. In *CCNC*. IEEE, 1–2.
[19] Mahmoud Nazzal, Abdallah Khreishah, Joyoung Lee, and Shaahin Angizi. 2023. Semi-decentralized Inference in Heterogeneous Graph Neural Networks for Traffic Demand Forecasting: An Edge-Computing Approach. *arXiv preprint arXiv:2303.00524* (2023).
[20] Siddhartha Sahu, Amine Mhedhbi, Semih Salihoglu, Jimmy Lin, and M Tamer Özsu. 2017. The ubiquity of large graphs and surprising challenges of graph processing. *Proceedings of the VLDB Endowment* 11, 4 (2017), 420–431.
[21] Ekaterina Tolstaya et al. 2020. Learning decentralized controllers for robot swarms with graph neural networks. In *Conference on robot learning*. PMLR, 671–682.
[22] Mingyu Yan, Lei Deng, Xing Hu, Ling Liang, Yujing Feng, Xiaochun Ye, Zhimin Zhang, Dongrui Fan, and Yuan Xie. 2020. Hygcn: A gcn accelerator with hybrid architecture. In *HPCA*. IEEE, 15–29.
[23] Mingyu Yan, Xing Hu, Shuangchen Li, Abanti Basak, Han Li, Xin Ma, Itir Akgun, et al. 2019. Alleviating irregularity in graph analytics acceleration: A hardware/software co-design approach. In *Micro*. 615–628.
[24] Da Zheng et al. 2020. Distdgl: distributed graph neural network training for billion-scale graphs. In *IA3*. IEEE, 36–44.
[25] Jie Zhou, Ganqu Cui, Shengding Hu, Zhengyan Zhang, Cheng Yang, Zhiyuan Liu, Lifeng Wang, Changcheng Li, and Maosong Sun. 2020. Graph neural networks: A review of methods and applications. *AI open* 1 (2020), 57–81.
[26] Zhenhua Zhu et al. 2020. MNSIM 2.0: A behavior-level modeling tool for memristor-based neuromorphic computing systems. In *GLSVLSI*. 83–88.