# Characterizing the Communication Requirements of GNN Accelerators: A Model-Based Approach

Robert Guirado, Akshay Jain, Sergi Abadal, and Eduard Alarcón
Universitat Politècnica de Catalunya

***Abstract*—Relational data present in real world graph representations demands for tools capable to study it accurately. In this regard Graph Neural Network (GNN) is a powerful tool, wherein various models for it have also been developed over the past decade. Recently, there has been a significant push towards creating accelerators that speed up the inference and training process of GNNs. These accelerators, however, do not delve into the impact of their dataflows on the overall data movement and, hence, on the communication requirements. In this paper, we formulate analytical models that capture the amount of data movement in the most recent GNN accelerator frameworks. Specifically, the proposed models capture the dataflows and hardware setup of these accelerator designs and expose their scalability characteristics for a set of hardware, GNN model and input graph parameters. Additionally, the proposed approach provides means for the comparative analysis of the vastly different GNN accelerators.**

***Index Terms*—GNN Accelerators, Parametric models**

## I. Introduction

Representation of data-sets as graph-based data structures, given the opportunities they provision to understand complex relationships embedded in them, has become increasingly popular [1]–[5]. These graph representations can range from very small (chemistry) to extremely huge (recommendation systems) graphs [6], [7]. Subsequently, to learn and infer from these graph representations, the seminal work by Scarselli *et al.* [8] in 2009 introduced the notion of Graph Neural Networks (GNNs). While multiple GNN models have been developed over the past decade [9]–[15], it is only recently that accelerators for speeding up the training and inference of GNNs have gained attention [16]–[22].

The area of GNN acceleration, however, is still in its infancy and the handful of accelerators that exist cover a wide portion of the design space, but only with few sparse points [5]. Moreover, as shown in Table I, these accelerators support a broad variety of GNN algorithm variants. While the accelerator designs verify their performance through metrics such as throughput and overall energy consumption, an explicit study on the data movement within the accelerators is absent. Note that, the data movement dictates the requirements cast on the underlying on-chip interconnect, which has a large impact on the latency and energy consumed by the accelerator. In fact, an important aspect of current GNN accelerator designs is to minimize data movement in order to be faster, more scalable, and more energy efficient. Hence, there is an interest on studying the effect of the dataflows and hardware design of the accelerators on the scalability trends of their underlying interconnect fabric [23]. This will also provision important insights which will help guide future GNN accelerator designs.

TABLE I: Selection of GNN accelerators in the literature.

| Accelerators | Algorithms Supported |
|---|---|
| EnGN [16] | GCN [9], GraphSage-Max [14], GatedGCN [11], GRN [16], R-GCN [15] |
| HyGCN [17] | GCN, GraphSage-Mean [14], GIN [24], DiffPool [12] |
| Auten *et al.* [19] | GCN, GAT [13], PGNN [19] |
| AWB-GCN [18] | GCN |
| GRIP [20] | GCN, GraphSage-Max, GIN, GatedGCN |

In this context, this paper makes the following contributions:

- It presents analytical models that describe the impact of the dataflows and hardware design of GNN accelerators on the overall data movement. The models are based on the accurate descriptions of the accelerator in question. To the best of our knowledge, this is the first work providing such analytical models, which will help in performing scalability analyses, comparing between different accelerator designs, and making more informed decisions with regards to future GNN accelerators.
- It evaluates two distinct state-of-the-art GNN accelerators, i.e., EnGN [16] and HyGCN [17], using the proposed models. This opens the door to the development of similar analytical models for other accelerators.

We provide background on GNNs in Sec. II, describe the models in Sec. III, discuss the scaling results in Sec. IV, and conclude the paper in Sec. V.

## II. GNN Accelerators: Background

GNN accelerators are application-specific integrated circuits that aim to process one or multiple GNN variants (see Table I) in a timely and energy efficient manner. The main challenge in GNN accelerator design is the alternation of phases with either dense or extremely sparse computation. The sparsity is driven by the graph connectivity or the graph adjacency matrix [16]–[18]. On the other hand, phases of dense computations are usually due to the dense nature of the operations that are applied to the nodes and edges in parallel [16]–[18]. Additionally, GNNs process input graphs that might have billions of nodes and edges, with uneven connectivity. Such characteristics lead to workload profiles that differ from conventional neural networks (NNs), besides being inherently imbalanced [18].

Consequently, multiple works [16]–[22] propose accelerators that speedup the inference/learning process with various specific architectural techniques. To illustrate the fundamental principles, Fig. 1 represents a generic architecture of a GNN accelerator with its corresponding inputs and outputs. Concretely, the acceleration engine takes as input a graph data

 DOI 10.1109/ISCAS51556.2021.9401612

structure and performs some initial pre-processing, such as tiling/graph partitioning [16]. This is then fed to the processing engine, which depending on its architecture and the predefined dataflow, processes the incoming graph partitions to yield an output. This output is generally a vector of features (i.e. predictions) for either a node, an edge, or the entire graph.

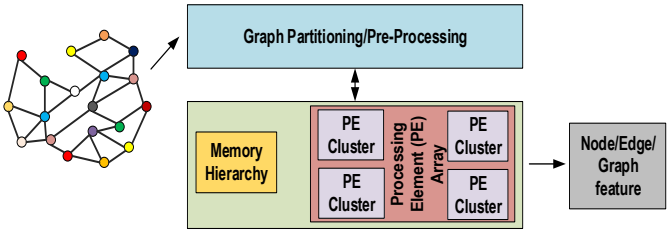
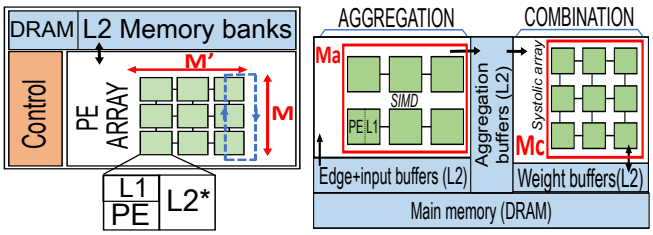



Fig. 1: Generic schematic diagram of a GNN accelerator.

The GNN inference process is generally divided into two computation stages, namely, aggregation and combination [5]. First, the data from nodes or edges is loaded from the memory hierarchy to the Processing Elements (PEs). In the aggregation stage, the node and/or edge features are added to that of their neighbours following the adjacency matrix, resulting into movement of data across PEs. Subsequently, in the combination stage, the aggregated features are transformed through a set of typical NN operations such as matrix-vector multiplications and a non-linear activation function (e.g., ReLU or tanh). Finally, after multiple recursions of the aforesaid stages, a readout of the desired features is performed via another NN or by just a linear/non-linear transformation. Training follows a similar process [5].

Given their distinct architectures, in this work we focus on two state-of-the-art GNN acceleration engines, i.e., EnGN [16] and HyGCN [17]. Figs. 2(a) and 2(b) illustrate their hardware architecture, which implement vastly differing methods for, among others, the graph partitioning and the reuse of vertex features and intermediate results. These design decisions, along with the specific dataflow they implement, impact the characteristics of data movement. Hence, we next describe analytical models of data movement for processing a single tile, that can help understand these characteristics.




(a) EnGN [16] (b) HyGCN [17]

Fig. 2: Schematic diagram of the analyzed GNN accelerators.

## III. Analytical Models for Data Movement

In this section, we describe the process towards obtaining the parametric models that characterize data movement in GNN accelerators. Table II lists the employed notation. In essence, we consider different input graph parameters, such as the number of nodes $K$ and edges $P$ in a tile, and different hardware parameters, such as the amount of differently-organized PEs $M, M', M_a, M_c$ and memory bandwidth $B$ [bits/iteration], to breakdown the necessary data movement in a given accelerator based on its defined dataflow.

The models determine the *amount of data movement*, this is, the total number of bits that must be moved between different memory hierarchy levels; and the *number of iterations*, this is, the number of steps required to move all that data due to PE or memory bandwidth constraints. High data movement will likely lead to high energy consumption, while many iterations may be an indicator of high latency. The hierarchy levels are also specified as they have an obvious impact on latency and energy. For instance, accessing a memory bank (L2) is $\sim 6\times$ more expensive than accessing a register file (L1) [25].

The proposed models are based on the analysis of the thorough architectural descriptions and walkthrough examples of EnGN [16] and HyGCN [17]. Validation of the data movement models is difficult as the authors of both accelerators provide metrics normalized to the CPU or GPU implementations mostly, and do not explicitly study data movement. Moreover, their simulation tools are in-house and not open source.

TABLE II: Summary of notation.

| Input graph parameters | | Architecture parameters | |
|---|---|---|---|
| N | Size of input feature vector | $\sigma$ | Bit precision |
| T | Size of output feature vector | B | L2 memory bandwidth |
| K | Number of vertices in a tile | M×M' | EnGN PE array size |
| L | Number of high-degree vertices in a tile | $M_a$<br>$M_c$ | HyGCN aggregation and combination PEs |
| P | Number of edges in a tile | $\Gamma$ | HyGCN systolic array reuse |
| | | $P_s$ | HyGCN edges after sliding |

Table III lists the amount of data movement and number of iterations in the different stages of EnGN. By default, this accelerator employs a single $128 \times 16$ PE array alongside multiple levels of memory to process both the aggregation and combination stages sequentially. First, a control engine determines how the input graph is streamed onto the processing engine. Vertices, both from L2 and a cache storing highly-connected vertices, start being loaded as stated in *loadvertL2* and *loadvertcache* respectively. We consider limitations in memory bandwidth $B$, and row PE array size $M$. Next, for the aggregation stage, a ring-edge-reduce (RER) whereby PEs send their outputs following a physical ring is proposed [16]. RER generates the data movement specified in *aggregate* due to its ring nature. After this, EnGN loads the weights for the combination stage, *loadweights*, we take into account the required amount of weights to process a tile, i.e., $N \times T$. Further, to compute the data that can be read in each iteration, we consider the minimum between $T$, the memory bandwidth $B$ and the compute array size. We chose the minimum of the aforesaid parameters since in the EnGN architecture vertex features are loaded on the PE engine in a streamed fashion. Finally, results are written in the cache and the L2 memory bank, *writecache* and *writeL2*, and the next tile is loaded, in *intertile*. The required iterations, as mentioned above, are then the result of a ceiling function between the total data movement and the data that can be moved at once.

TABLE III: EnGN analytical model. L2* refers to a dedicated vertices cache.

| Movement level | Data movement | Number of iterations | Hierarchy |
|---|---|---|---|
| loadvertcache | min(L$\sigma$,M$\sigma$,B*)·N·ceil(L$\sigma$/min(B*,M$\sigma$)) | ceil(L$\sigma$/min(B*,M$\sigma$)) | L2*–L1 |
| loadvertL2 | min((K–L)$\sigma$,M$\sigma$,B)·N·ceil((K-L)$\sigma$/min(B,M$\sigma$)) | ceil((K-L)$\sigma$/min(B,M$\sigma$)) | L2–L1 |
| loadedges | min(P$\sigma$,B)·ceil(P$\sigma$/B) | ceil(P$\sigma$/B) | L2–L1 |
| loadweights | min(T$\sigma$,M$\sigma$,B)·N·ceil(T$\sigma$/min(B,M$\sigma$)) | ceil(T$\sigma$/min(B,M$\sigma$)) | L2–L1 |
| aggregate | M(M-1)T·(ceil(K/M)+ceil(K(N-M)/M))$\sigma$ | ceil(K/M)+ceil(K(N-M)/M) | L1–L1 |
| writecache | min(M$\sigma$,L$\sigma$, B*)T·ceil(L$\sigma$/min(M$\sigma$,B*)) | ceil(L$\sigma$/min(M$\sigma$,B*)) | L1–L2 |
| writeL2 | min(M$\sigma$,(K-L)$\sigma$, B)T·ceil((K-L)$\sigma$/min(M$\sigma$,B)) | ceil((K-L)$\sigma$/min(M$\sigma$,B)) | L1–L2 |

TABLE IV: HyGCN analytical model.

| Movement level | Data movement | Number of iterations | Hierarchy |
|---|---|---|---|
| loadvertL2 | min(K$\sigma$,$M_a\sigma$,B))·N·ceil(K$\sigma$/min(B,$M_a\sigma$) | ceil(K$\sigma$/min(B,$M_a\sigma$)) | L2–L1 |
| loadedges | min($P_s\sigma$,B)ceil($P_s\sigma$/B) | ceil($P_s\sigma$/B) | L2–L1 |
| loadweights | min(NT$\sigma(1-\Gamma)$,$M_c\sigma$,B)ceil(NT$\sigma(1-\Gamma)$/min(B, $M_c\sigma$)) | ceil(NT$\sigma(1-\Gamma)$/min(B, $M_c\sigma$)) | L2–L1 |
| aggregate | min($NP_s\sigma$,$M_a$8)ceil($NP_s\sigma$/($M_a$8)) | ceil($NP_s\sigma$/($M_a$8)) | L1–L1 |
| writeinterphase | min(KN$\sigma$, B)ceil(KN$\sigma$/B) | ceil(KN$\sigma$/B) | L1–L2 |
| combine | KN$\sigma$+ NT$\sigma$ | 1 | L1–L1 |
| readinterphase | min($P_s$N$\sigma$,B,$M_c$)ceil($P_s$N$\sigma$/min(B,$M_c$)) | ceil($P_s$N$\sigma$/min(B,$M_c$)) | L2–L1 |
| writeL2 | min(KT$\sigma$, B)ceil(KT$\sigma$/B) | ceil(KT$\sigma$/B) | L1–L2 |

Table IV, on the other hand, presents the model for HyGCN. HyGCN consists of two separate engines for the pipelined execution of the aggregation (PE array with 32 Single-Input-Multiple-Data cores) and combination (a systolic array with $8\times4\times128$ PEs). The memory organization is slightly different than in EnGN, using an extra aggregation buffer to store intermediate results [17]. To account for these particularities, we introduce a $\Gamma$ parameter, which reflects the reusability of data within the combination engine, i.e., the systolic array. To process a tile in HyGCN, graph data is loaded from L2 input and edge buffers, *loadvertL2* and *loadedges*. The aggregation task, *aggregation*, is handled by all vertices at the same time, each of them working on up to eight separate feature components. Then, aggregated features are stored in the inter-phase cache, ready to be fetched by the combination engine, *writeinterphase* and *readinterphase*, and proceed with the matrix-vector multiplications. The only remaining step is to write the results to the output buffer, *writeL2*.

## IV. Results and Discussion

We evaluate the total data movement, in bits, and number of iterations for the processing of a tile, i.e., fixed-size portion of a graph. One could extend the analysis to arbitrary graphs by multiplying by its number of tiles and modeling the reuse across tiles. Here, we sweep both input (number of vertices $K$) and architectural parameters (number of PEs and bandwidth) to obtain scalability trends. Other parameters have default values of $N = 30$, $T = 5$, $B = 1000$, $\sigma = 4$. The number of edges is $P = 10 \cdot K$ to model similar connectivity in growing tiles. We set $P_s \sim P$ as the performance of HyGCN's sliding window may vary across tiles.

### A. EnGN

Fig. 3 shows the data movement in an EnGN-like accelerator, for different graph sizes and PE array size. We take $M = M'$ for the sake of clarity. It can be observed how aggregation dominates and leads to over $10\times$ more data movement than *loadvertL2*. This is due to the RER-based aggregation strategy. However, since this movement is happening between L1 memory levels, it is faster and less energy-consuming than other data paths. Moreover, we can observe how data movement increases linearly with $K$, but not with $M \times M'$. In the latter case, small PE arrays struggle to compute the tile efficiently as they need to continuously fetch new values into the RER, which increases data movement. Another interesting result is that *loadvertcache* data, fetched from the dedicated cache for highly-connected vertices, relieves significant load from the vertex memory bank at *loadvertL2*. As compared to HyGCN, EnGN shows a smaller *loadvertL2*, since high-degree vertices are handled by the faster and closer cache memory.

Next, Fig. 5(a) shows how the total number of iterations scale with the memory bandwidth $B$, which is useful to detect bandwidth over-provisioning regions. We observe that bandwidth makes a bigger difference in smaller tiles as data is constantly moved in and out of the PE array. This is seen through both the saturation point and the value at which the curve saturates.

We finally note that previous results, especially from Fig. 3, seem to suggest that there is an optimal PE array size that depends on the tile size. This is because EnGN tries to fit the tiles in its PE array of size $M^2$ if $M = M'$, which is used both for aggregation and combination. Since a tile has a total size of $K$ nodes multiplied by the $N$ input features, an array fitting factor K·N/M$^2$ has also been studied. By sweeping this factor as shown in Fig. 6, it is observed that whenever the number of elements to be placed in the array is large enough so that the number of PEs cannot host them, the number of iterations starts increasing since the aggregation and combination will take several steps to complete. This would help to explain why in Fig. 3 data movement first decreases and then increases with $M$, thus shows the potential of this methodology to reveal accelerator-specific behaviors.

### B. HyGCN

Fig. 4 breaks down the total data movement of HyGCN. By comparing to Fig. 3, we can provide interesting insights. First, we observe that data movement increases almost linearly with the tile size, but is independent of the array size. Second, HyGCN involves moves significantly more data than EnGN, due to its dual architecture and the need to write-read from the aggregation buffer. Moreover, HyGCN's aggregation depends

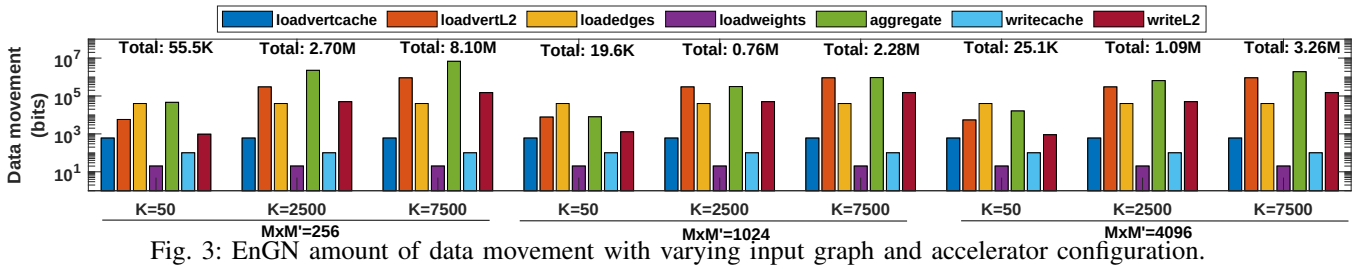



Fig. 3: EnGN amount of data movement with varying input graph and accelerator configuration.

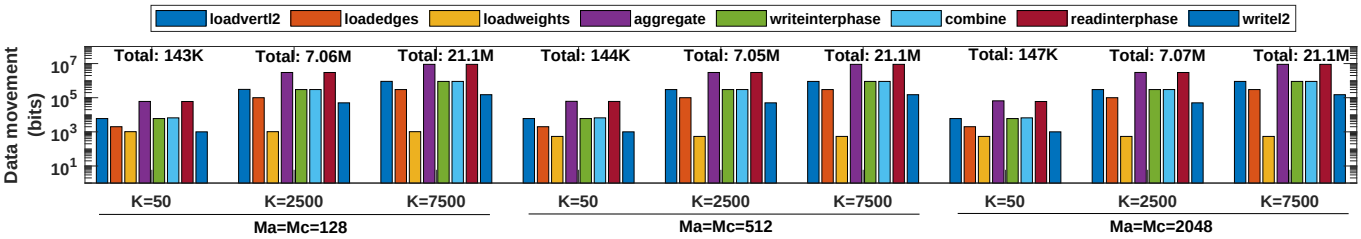



Fig. 4: HyGCN amount of data movement with varying input graph and accelerator configuration.

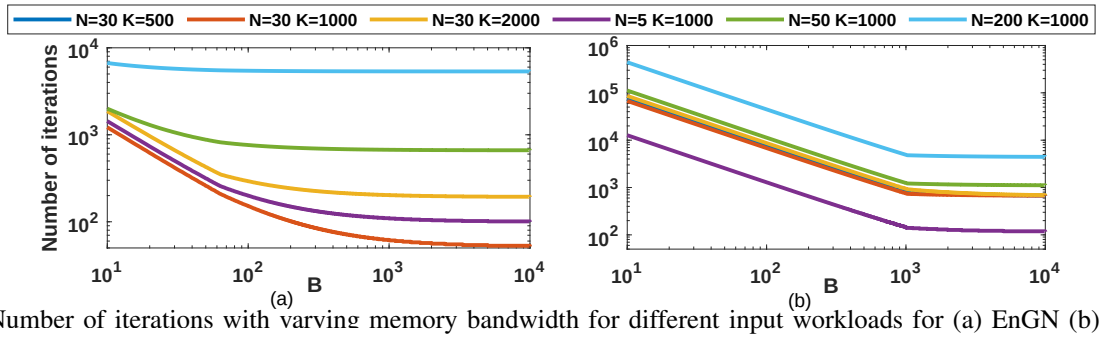



Fig. 5: Number of iterations with varying memory bandwidth for different input workloads for (a) EnGN (b) HyGCN.

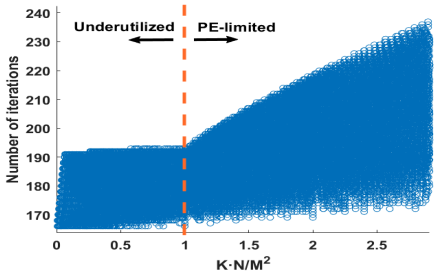



Fig. 6: EnGN number of iterations of data movement with varying array fitting factor.

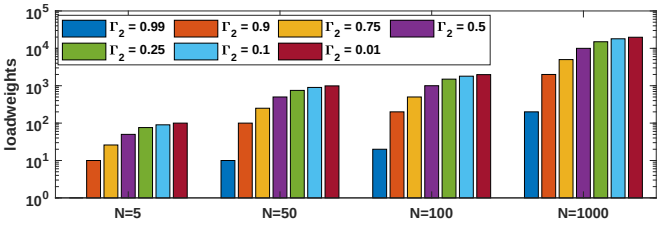



Fig. 7: HyGCN *loadweights* data movement with varying systolic array reuse.

on the size of the input feature vector $N$ instead of the output vector $T$, leading to large aggregations and phase transitions.

Further, Fig. 5(b) represents the HyGCN analogous to Fig. 5(a). The figure illustrates how HyGCN is also sensitive to bandwidth and how the saturation point, unlike in EnGN, is abrupt and independent on the tile size. This is mostly the effect of having aggregation buffers to orchestrate movement across phases, which makes HyGCN more bandwidth-hungry.

In order to model HyGCN-specific features, Fig. 7 shows the effect of the systolic array reuse $\Gamma$ over *loadweights*, for different graph depths $N$. We can see how large values of $\Gamma$, representing large reuse, alleviate the amount of weights to be loaded. This highlights the importance of tiling.

## V. Conclusion

We have presented a characterization method of the data movement within GNN accelerators based on analytical models. As a precursor of communication requirements, the proposed method can help comparing different architectures and explore their design spaces. In this work, we have tested the proposed method on two novel GNN accelerators and observed that (i) aggregation is accountable for a large fraction of data movement, (ii) the two architectures scale very differently, and (iii) memory limitations lead to an increase of total iterations, affecting latency. As future work, the analytical models for these and other accelerators will be expanded to model graph properties such as sparsity and validated against cycle-accurate simulations with dedicated tools.

## VI. Acknowledgements

The authors gratefully acknowledge support from the Spanish Ministry of Science and Innovation (MICINN) under grant EIN2019-103461 and from the European Comission under grant H2020-863337-WIPLASH.

## REFERENCES


[1] M. M. Bronstein, J. Bruna, Y. Lecun, A. Szlam, and P. Vandergheynst, "Geometric Deep Learning: Going beyond Euclidean data," *IEEE Signal Process. Mag.*, vol. 34, no. 4, pp. 18–42, 2017.

[2] P. W. Battaglia *et al.*, "Relational inductive biases, deep learning, and graph networks," *arXiv preprint arXiv:1806.01261*, 2018.

[3] Z. Zhang, P. Cui, and W. Zhu, "Deep Learning on Graphs: A Survey," *IEEE Trans. Knowl. Data Eng.*, 2020.

[4] M. Eisen and A. R. Ribeiro, "Optimal wireless resource allocation with random edge graph neural networks," *IEEE Transactions on Signal Processing*, 2020.

[5] S. Abadal, A. Jain, R. Guirado, J. López-Alonso, and E. Alarcón, "Computing graph neural networks: A survey from algorithms to accelerators," *arXiv preprint arXiv:2010.00130*, 2020.

[6] W. Fan, Y. Ma, Q. Li, Y. He, E. Zhao, J. Tang, and D. Yin, "Graph neural networks for social recommendation," *Proceedings of the WWW'19*, pp. 417–426, 2019.

[7] A. Fout, J. Byrd, B. Shariat, and A. Ben-Hur, "Protein interface prediction using graph convolutional networks," *Proceedings of the NIPS'17*, pp. 6531–6540, 2017.

[8] F. Scarselli, M. Gori, Ah Chung Tsoi, M. Hagenbuchner, and G. Monfardini, "The Graph Neural Network Model," *IEEE Trans. Neural Networks*, vol. 20, no. 1, pp. 61–80, 2009.

[9] T. N. Kipf and M. Welling, "Variational graph auto-encoders," *arXiv preprint arXiv:1611.07308*, 2016.

[10] T. N. Kipf and M. Welling, "Semi-supervised classification with graph convolutional networks," *Proceedings of the ICLR 2017*, sep 2017.

[11] X. Bresson and T. Laurent, "Residual gated graph convnets," *arXiv preprint arXiv:1711.07553*, 2017.

[12] Z. Ying, J. You, C. Morris, X. Ren, W. Hamilton, and J. Leskovec, "Hierarchical graph representation learning with differentiable pooling," in *Proceedings of the NIPS'18*, pp. 4800–4810, 2018.

[13] V. P. Dwivedi, C. K. Joshi, T. Laurent, Y. Bengio, and X. Bresson, "Benchmarking graph neural networks," *arXiv preprint arXiv:2003.00982*, 2020.

[14] W. L. Hamilton, R. Ying, and J. Leskovec, "Inductive representation learning on large graphs," *Proceedings of NIPS'17*, pp. 1025–35, 2017.

[15] M. Schlichtkrull, T. N. Kipf, P. Bloem, R. van den Berg, I. Titov, and M. Welling, "Modeling Relational Data with Graph Convolutional Networks," *Proceedings of the ESWC 2018*, pp. 593–607, 2018.

[16] S. Liang, Y. Wang, C. Liu, L. He, L. Huawei, D. Xu, and X. Li, "EnGN: a high-throughput and energy-efficient accelerator for large graph neural networks," *IEEE Transactions on Computers*, 2020.

[17] M. Yan, L. Deng, X. Hu, L. Liang, Y. Feng, X. Ye, Z. Zhang, D. Fan, and Y. Xie, "HyGCN: a GCN accelerator with hybrid architecture," in *Proceedings of the HPCA-26*, pp. 15–29, 2020.

[18] T. Geng, A. Li, R. Shi, C. Wu, T. Wang, Y. Li, P. Haghi, A. Tumeo, S. Che, S. Reinhardt, *et al.*, "AWB-GCN: a graph convolutional network accelerator with runtime workload rebalancing," in *Proceedings of the MICRO-53*, 2020.

[19] A. Auten, M. Tomei, and R. Kumar, "Hardware Acceleration of Graph Neural Networks," *Proceedings of the DAC'20*, 2020.

[20] K. Kiningham, C. Re, and P. Levis, "GRIP: A Graph Neural Network Accelerator Architecture," *arXiv preprint arXiv:2007.13828*, 2020.

[21] B. Zhang, H. Zeng, and V. Prasanna, "Hardware Acceleration of Large Scale GCN Inference," in *Proceedings of the ASAP '20*, pp. 61–68, 2020.

[22] H. Zeng and V. Prasanna, "GraphACT: Accelerating GCN training on CPU-FPGA heterogeneous platforms," *Proceedings of the FPGA 2020*, pp. 255–265, 2020.

[23] R. Guirado, H. Kwon, E. Alarcón, S. Abadal, and T. Krishna, "Understanding the Impact of On-chip Communication on DNN Accelerator Performance," in *2019 26th IEEE International Conference on Electronics, Circuits and Systems (ICECS)*, pp. 85–88, 2019.

[24] K. Xu, S. Jegelka, W. Hu, and J. Leskovec, "How powerful are graph neural networks?," *Proceedings of the ICLR 2019*, pp. 1–17, 2019.

[25] Y.-H. Chen, J. Emer, and V. Sze, "Eyeriss: A Spatial Architecture for Energy-Efficient Dataflow for Convolutional Neural Networks," in *Proceedings of the ISCA-43*, p. 367–379, 2016.