

# G-CoS: GNN-Accelerator Co-Search Towards Both Better Accuracy and Efficiency

Yongan Zhang<sup>1</sup>, Haoran You<sup>1</sup>, Yonggan Fu<sup>1</sup>, Tong Geng<sup>2</sup>, Ang Li<sup>2</sup>, Yingyan Lin<sup>1</sup>

<sup>1</sup>Rice University, Houston, TX; <sup>2</sup>Pacific Northwest National Laboratory, Richland, WA

Corresponding to: <sup>1</sup>yingyan.lin@rice.edu

**Abstract**—Graph Neural Networks (GNNs) have emerged as the state-of-the-art (SOTA) method for graph-based learning tasks. However, it still remains prohibitively challenging to inference GNNs over large graph datasets, limiting their application to large-scale real-world tasks. While end-to-end jointly optimizing GNNs and their accelerators is promising in boosting GNNs' inference efficiency and expediting the design process, it is still underexplored due to the vast and distinct design spaces of GNNs and their accelerators. In this work, we propose G-CoS, a GNN and accelerator co-search framework that can automatically search for matched GNN structures and accelerators to maximize both task accuracy and acceleration efficiency. Specifically, G-CoS integrates two major enabling components: (1) a *generic GNN accelerator search space* which is applicable to various GNN structures and (2) a *one-shot GNN and accelerator co-search algorithm* that enables simultaneous and efficient search for optimal GNN structures and their matched accelerators. To the best of our knowledge, G-CoS is the first co-search framework for GNNs and their accelerators. Extensive experiments and ablation studies show that the GNNs and accelerators generated by G-CoS consistently outperform SOTA GNNs and GNN accelerators in terms of both task accuracy and hardware efficiency, while only requiring a few hours for the end-to-end generation of the best matched GNNs and their accelerators.

**Index Terms**—GNN-Accelerator, Neural Architecture Search

## I. INTRODUCTION

Graph neural networks (GNNs) [1] have gained an increased popularity recently as they demonstrated the state-of-the-art (SOTA) performance in various graph-based learning tasks, including node classification [1], graph classification [2], and recommendation systems [3]. However, GNNs often suffer from prohibitive inference cost, limiting their potential to handle large-scale real-world graph applications. For example, a 2-layer GNN model requires 19G FLOPs (FLOPs: floating point operations) to inference the Reddit graph [4], which requires a latency of 2.94E+5 milliseconds when being executed on an Intel Xeon E5-2680 CPU platform [5], i.e., its required FLOPs and latency are 2× and 5000× of a 50-layer convolutional neural network (CNN), ResNet-50 [6].

The giant computational cost of GNN inference results from three aspects. First, graphs are often very large as exacerbated by their intertwined complex neighbor connections, e.g., a total of 232,965 nodes in the Reddit graph with each node having about 50 neighbors [7]. Second, real-world graphs tend to follow the power-law distribution and therefore have highly irregular adjacent matrices, resulting in prohibitive inefficiencies in both data processing and movements. Third, the dimension of GNNs' node feature vectors can be very high, e.g., each node in the CiteSeer graph has 3703 features.

To tackle GNNs' prohibitive inference cost, various efficient GNN inference techniques have been developed. On the algorithm level, several pioneering GNN compression techniques have been developed. For instance, two concurrent GNN pruning works [8], [9] aim to sparsify the connections in GNNs' graph adjacent matrices; and [4] for the first time shows the feasibility of adopting 8-bit integer arithmetic for GNN inference without sacrificing the accuracy. Another paralleled trend is to search for efficient GNN architectures [10], [11]. On the hardware level, various GNN accelerators have been proposed. For example, HyGCN [12] proposes hybrid execution patterns of GNNs for leveraging their intra-vertex and inter-vertex parallelisms to handle the irregularity in the aggregation phase and reusability in the combination phase, respectively; Later, AWB-GCN [5] identifies the workload imbalance problem in the aggregation phase, and proposes auto-tuning workload balancing techniques, achieving an average speedup of 7.4× over HyGCN. On the development tool level, pioneering works have attempted to characterize the design space of dataflows and micro-architectures for GNN accelerators [13], and develop an automated framework to generate GNN accelerators [14].

Despite the promising performance of existing efficient GNN inference solutions, their achievable efficiency is still not sufficient for enabling extensive GNN inference applications due to GNNs' extremely dynamic and irregular data accesses and thus excessive acceleration cost. Motivated by the great success of algorithm-accelerator co-exploration works for CNN accelerations [15]–[21], this work targets to co-optimize both the GNN structures and their accelerators with boosted development speed, and makes the following contributions:

- We propose G-CoS, a GNN and accelerator co-search framework that can automatically search for the matched GNN structures and accelerators to maximize both task accuracy and acceleration efficiency. To the best of our knowledge, G-CoS is the first co-search framework for GNNs and their accelerators.
- G-CoS integrates two enabling components: (1) a *generic GNN accelerator search space* which is applicable to various GNN structures and (2) a *one-shot GNN and accelerator co-search algorithm* that enables simultaneous and efficient search for optimal GNN structures and their matched accelerators, both of which can facilitate the algorithmic exploration of efficient GNN solutions.
- Extensive hardware/algorithm experiments and ablation

studies validate G-CoS's effectiveness and advantage: G-CoS generated networks/accelerators consistently outperform SOTA GNNs/accelerators, while requiring only a few hours for the end-to-end search, (e.g., 4 GPU hours for the Cora dataset).

## II. RELATED WORKS

**Graph neural networks (GNNs).** GNNs have achieved great success in graph-based learning tasks [22], [23]. Depending on their graph representation domains, GNNs can be categorized into spectral and spatial GNNs. Specifically, spectral GNNs model the representation in the graph Fourier transform domain based on eigen-decomposition and usually handle the whole graph simultaneously [24], [25]. However, it becomes impractical for spectral GNNs to process large graphs and difficult for them to take advantage of parallel processing [10], [26]. On the other hand, spatial GNNs [7], [27], which directly perform the computation in the graph domain by aggregating the neighbor nodes' features, have undergone rapid development. Recently, [28] introduced an attention mechanism to further improve the performance of spatial GNNs; and [29] utilized mini-batch training to improve GNNs' scalability to handling large graphs. Combined with sampling strategies, the whole graph is no longer required during aggregation, leaving much room for potential hardware acceleration [7], [30]. Therefore, in this work, we will primarily focus on the spatial GNNs for their advantages on scalability and potential hardware acceleration.

**Graph neural architecture search (GNAS).** Neural architecture search (NAS) has become a popular approach to designing neural networks [31]–[33], which can significantly relieve human efforts from manually designing complex networks. The recent NAS success and the large distinction among GNN structures for different tasks have motivated the use of NAS for GNNs (denoted as GNAS). For example, [10], [34] used reinforcement learning (RL) methods along with parameter sharing to efficiently search for GNNs; [35] adopted an evolutionary search algorithm; and [36] proposed a more generic GNN design space and a standardized evaluation method for GNNs across various graph learning tasks. Despite the preliminary success, existing works still heavily rely on excessive rounds of sampling and retraining, limiting their scalability to more generic search spaces.

**Hardware-aware architecture search (HA-NAS).** To ensure the searched networks' hardware efficiency, hardware-aware NAS (HA-NAS) proposes to incorporate hardware metrics, e.g., the latency on mobile phones, into the search process. Early works, e.g., [37]–[39], utilized RL-based methods, and thus suffered from substantial search time and costs, limiting their scalability to larger and more diverse search spaces. Inspired by DARTS [40], differentiable HA-NAS [21], [41]–[43] has emerged to greatly improve both the search and hardware efficiency. However, restricted by the large difference among different GNN structures and thus the difficulty for hardware acceleration, HA-NAS targeting GNNs has rarely been explored. Furthermore, existing HA-NAS methods have

not yet fully explored the hardware design space. As the acceleration efficiency is determined by both the network structures and their accelerators, it is thus desirable to jointly search for both the networks and their accelerators.

**GNN inference accelerators.** GNNs' ultra-sparse graph matrices, corresponding to extremely dynamic and irregular data accesses as well as distinct execution patterns from DNNs, have fueled a growing interest in developing dedicated GNN accelerators [44]. For instance, HyGCN [12] explored both intra/inter-vertex parallelisms to separately handle the irregularity in the aggregation phase and reusability in the combination phase. Later, aiming to boost the overall hardware utilization, AWB-GCN [5] proposed to balance the workload during runtime with an auto-tuning algorithm and to increase the data locality by regionally clustering the non-zero values (i.e., connected neighbors) within the adjacency matrices; EnGN [45] proposed a ring-edge-reduce dataflow to handle graphs with arbitrary dimensions and increase the accelerator's scalability to large graphs; and GRIP [46] employed fine-grained vertex-tiling to reduce the weight bandwidth requirements; In parallel, to reduce the human efforts in designing GNN accelerators and democratize the process, pioneering works have attempted to characterize the design space of dataflows and micro-architectures for GNN accelerators [13], and developed an automated framework to generate GNN accelerators [14]. Nevertheless, existing automated frameworks for GNNs still have limited support to various GNN structures and thus suffer from low hardware utilization and achievable efficiency on certain tasks.

**Software/Hardware Co-exploration.** Jointly exploring the networks and their accelerators has shown promising results [15]–[21], [47]. For instance, [16], [20] conducted RL-based search to jointly optimize the networks and some design parameters of FPGA-based accelerators; [15] developed the first differentiable network and accelerator co-search framework to boost both the task accuracy and acceleration efficiency; and [18] co-searches for networks, bitwidths, and accelerators to achieve superior performance. However, co-optimizing the GNN structures and their accelerators has not been studied.

## III. GNN PRELIMINARIES

### A. GNN notation and formulation

A typical GNN graph can be represented as,  $G = (V, E)$ , where  $v_i \in V$  and  $(v_i, v_j) \in E$  denote the nodes and edges, respectively; and  $N = |V|$  and  $M = |E|$  denote the total number of nodes and edges, respectively. The node degree is denoted as  $d = \{d_1, d_2, \dots, d_N\}$  where  $d_i$  indicates the number of neighbors connected to node  $v_i$ . We define  $D$  as the degree matrix whose diagonal elements are formed using  $d$ . The connectivity information is encoded within the adjacency matrix  $A \in \mathbb{R}^{N \times N}$ , where the non-zero entries represent the existed connections among different nodes. For each layer  $l$  of a GNN, the nodes are encoded by their feature vectors  $\{x_1^{(l)}, x_2^{(l)}, \dots, x_N^{(l)}\} = X^{(l)}$ , where  $X^{(l)} \in \mathbb{R}^{N \times F^{(l)}}$  and  $F^{(l)}$

denotes the feature dimension used to encode the nodes at layer  $l$ . Thus, a GNN layer [24] can be formulated as:

$$X^{(l+1)} = \text{ACT}_{(l)} \left( \hat{A} X^{(l)} W^{(l)} \right), \quad (1)$$

where  $\hat{A}$  is a normalized version of  $A$ :  $\hat{A} = D^{-\frac{1}{2}} A D^{-\frac{1}{2}}$ ,  $\text{ACT}_{(l)}$  represents the activation function of layer  $l$  and  $W^{(l)} \in \mathbb{R}^{F_{(l)} \times K_{(l)}}$  represents the weights in layer  $l$ , with  $K_{(l)}$  denoting the hidden/weight dimension at layer  $l$ . The whole GNN inference can thus be viewed as two separated phases: *Aggregation* and *Combination*.

- *Aggregation*  $\hat{A} X^{(l)}$ : For each node in the graph, a GNN aggregates its 1-hop neighbor nodes' features into a unified feature vector, corresponding to the multiplication between the adjacent and feature matrix, i.e.,  $\hat{A} X$ .
- *Combination*  $[\hat{A} X^{(l)}] W^{(l)}$ : The aggregated feature vector,  $\hat{A} X^{(l)}$ , will be further transformed to another feature vector via an MLP network (shared among nodes) with weights  $W^{(l)}$  for learning better representations, corresponding to the multiplication between the aggregated feature matrix and weight matrix, i.e.,  $[\hat{A} X^{(l)}] W^{(l)}$ .

In the final/prediction layer of GNNs, after the feature vectors' update, a softmax function is usually applied in a row-wise manner, i.e.,  $\text{softmax}(x_i^{(l)}) = \exp(x_i^{(l)}) / \sum_i \exp(x_i^{(l)})$  [24]. For semi-supervised multiclass classification, the loss function captures the cross-entropy errors over all labeled examples:

$$\mathcal{L}_{GNN}(W) = - \sum_{n \in \mathcal{Y}_N} \sum_f Y_{nf} \ln(\Theta_{nf}), \quad (2)$$

where  $\mathcal{Y}_N$  is the set of node indices that have labels,  $Y_{nf}$  is the ground truth label matrix, and  $\Theta_{nf}$  denotes the predicted possibilities of node  $n$  belonging to class  $f$ . During GNN training,  $W^{(l)}$  is updated via gradient descents.

### B. GNN variants and their implementations

Many advanced GNN variants have recently been proposed to consider different aggregation functions and introduce additional attention modules or sampling functions. Without loss of generality, we summarize four popular GNN architectures: **GCN** [24], **GAT** [28], **GIN** [48], and **GraphSAGE** [7]. We analyze the difference among them as compared with vanilla GNNs below, **aiming to generally support them in G-CoS**.

**GCN** [24]: During inference, each node can be written as  $x_i^{(l+1)} = \sum_{j \in \mathcal{N}(i) \cup i} (\frac{1}{d_i d_j} W^{(l)} x_j^{(l)})$ , where  $l$  is the layer index and  $\mathcal{N}(i)$  represents the  $i$ -th node's neighbor set. Thus, compared with the vanilla GNNs, the only difference lies in the entries of the adjacency matrix where each node is encoded as  $\frac{1}{d_i d_j}$  and can be filled offline before the processing.

**GAT** [28]: An attention module is introduced, i.e.,  $x_i^{(l+1)} = \alpha_{i,i} W^{(l)} x_i^{(l)} + \sum_{j \in \mathcal{N}(i)} (\alpha_{i,j} W^{(l)} x_j^{(l)})$ , where  $\alpha$  denotes the attention coefficients for the neighbor nodes and can be viewed as the elements to replace the original adjacency matrix's

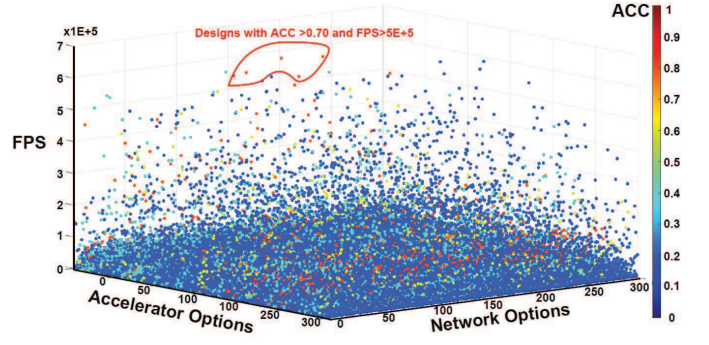


Fig. 1. FPGA measured Frame-Per-Second (FPS; see the left axis) on a VCU128 FPGA [50] and accuracy on Cora dataset (see the right colorbar) of 300 randomly sampled fully trained GNNs from supernet as defined in Sec. IV-D, when each of the networks is accelerated by 300 randomly sampled accelerators from the accelerator design space (see Sec. IV-C), leading to a total of 9E+4 randomly sampled data points. Designs with  $ACC > 0.7$  and  $FPS > 5E+5$  are circled out in red, which are extremely sparse.

entries. Adapting from the formulation of GAT [28], [49],  $\alpha_{i,j}$  can be calculated as:

$$\alpha_{i,j} = \frac{\exp(\text{ACT}(x_i^T w_1^{(l)} + x_j^T w_2^{(l)}))}{\sum_{k \in \mathcal{N}(i) \cup i} \exp(\text{ACT}(x_i^T w_1^{(l)} + x_k^T w_2^{(l)}))} \quad (3)$$

where  $\text{ACT}$  denotes the activation used in the attention module and  $(w_1^{(l)}, w_2^{(l)}) \in (\mathbb{R}^{F_{(l)} \times 1}, \mathbb{R}^{F_{(l)} \times 1})$  denotes the weights of the attention module. The whole set of  $\alpha$  can then be calculated by introducing an additional layer of matrix multiplication of  $X^{(l)} * [w_1^{(l)} || w_2^{(l)}]$  along with the element-wise activation and multiplication with the original adjacency matrix. Thus, *replacing the original adjacency matrix with  $\alpha$*  captures the functionality of the attention module.

**GIN** [48]: An information-lossless aggregation function is adopted, i.e.,  $x_i^{(l+1)} = \text{MLP}((1+\epsilon)x_i^{(l)} + \sum_{j \in \mathcal{N}(i)} x_j^{(l)})$ , where  $\text{MLP}$  denotes an MLP network and  $\epsilon$  is a learnable constant. As  $\epsilon$  is trained and then fixed during inference, GIN can be realized by *fusing  $\epsilon$  into the original adjacency matrix and incorporating an additional MLP layer into the aggregation phase of vanilla GNNs*.

**GraphSAGE** [7]: Uniform neighbor sampling is introduced to alleviate the extreme memory consumption during training, i.e.,  $x_i^{(l+1)} = \text{Mean}(W^{(l)} x_j^{(l)})$ ,  $j \in \{i\} \cup \{\mathcal{S}(i)\}$ , where  $\mathcal{S}(i)$  denotes the sampled neighbors, which can be easily supported by *introducing an additional node sampling layer and using mean aggregation* to the original GNN formulation as in Eq. 1.

## IV. THE PROPOSED G-CoS FRAMEWORK

This section describes our G-CoS framework by first providing an overview and the problem formulation, and then G-CoS's two major enablers: a generic GNN accelerator design space and network structure design space, followed by G-CoS's efficient one-shot evolutionary co-search algorithm.

### A. G-CoS: the overview and challenges

To maximize both task accuracy and hardware efficiency, G-CoS jointly searches for the best matched GNN structures and accelerators, under the specified datasets, resource constraints,



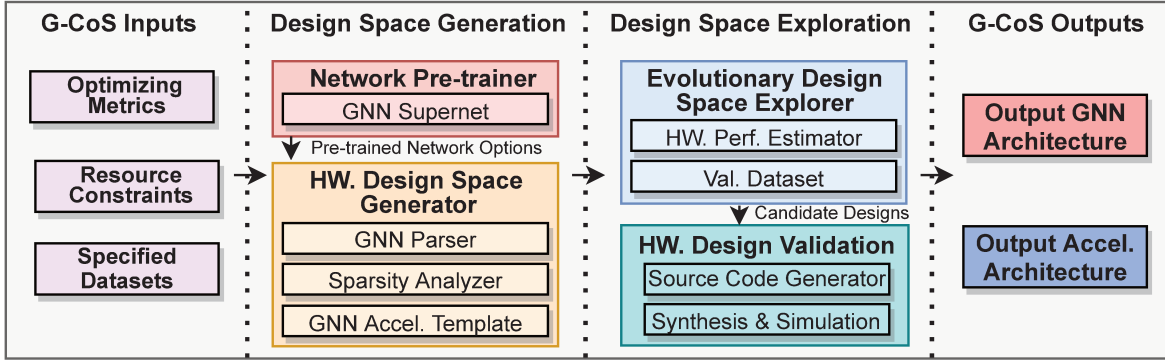


Fig. 2. An overview of our G-CoS GNN-accelerator co-search framework, where Accel. denotes accelerators.

and optimizing metrics (e.g., accuracy and latency), as shown in Fig. 2.

To enable effective GNN-accelerator co-search, there exist **three major challenges**, including (1) the prohibitively large and distinct joint space versus very sparse optima excelling at both accuracy and efficiency, as shown in Fig. 1, (2) the excessive retraining cost during GNAS, and (3) the lack of either generic GNN structure or accelerator search space description. To tackle the first two aforementioned challenges, G-CoS employs a *one-shot evolutionary GNN-accelerator co-search algorithm*, as introduced in Sec. IV-E. G-CoS first one-shot pre-trains the proposed GNN supernet to avoid the necessity of cumbersome retraining in the GNN-accelerator co-search phase, and then utilizes an evolutionary search algorithm to efficiently navigate the joint network-accelerator space to locate the optimal GNN-accelerator pairs based on the feedback of the estimated inference accuracy and hardware efficiency. For the third challenge, G-CoS integrates (1) a *generic GNN network space* description which is compatible with its one-shot search algorithm and (2) a *generic GNN accelerator design space* which includes accelerators with high hardware utilization across various GNN structures.

#### B. G-CoS: the co-optimization formulation

G-CoS's co-optimization process can be formulated as:

$$\arg \min_{\{gnet, hw\}} L_{val}(\omega^*, gnet, hw) + \lambda L_{cost}(gnet, hw) \quad (4)$$

$$s.t. \quad \omega^* = \arg \min_{\omega} L_{train}(\omega, gnet), \quad (5)$$

where  $\omega$  denotes the GNN weights;  $L_{train}$ ,  $L_{val}$ , and  $L_{cost}$  are the task loss during training, task loss during validation, and hardware-cost, respectively, given the GNN structure, the accelerator parameter set, and the specified metrics; and  $gnet$  and  $hw$  are the selected GNN structure and accelerator to be optimized, respectively. Note that the hardware-cost is co-determined by both the GNN structure and accelerator.

#### C. G-CoS: a generic GNN accelerator template and space

To comprehensively cover potential parallelism and reuse opportunities for various GNN structures, we propose a generic GNN accelerator template along with a set of corresponding searchable parameters, forming a design space featuring a total of  $\sim 1E+15$  GNN accelerator choices with different micro-architectures and dataflows.

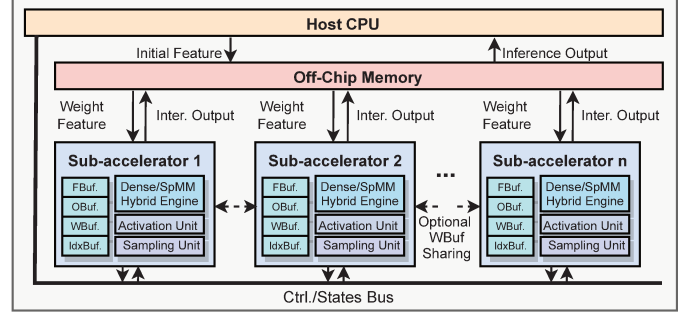


Fig. 3. An illustration of G-CoS's accelerator micro-architecture template.

**The micro-architecture overview.** To maximize the acceleration throughput while at the same time minimizing the latency, we adopt a multi-accelerator micro-architecture template to accelerate both the combination and aggregation phases, as shown in Fig. 3. This template has two overall advantages:

- **Latency friendly:** When working on either of the two phases, all the hardware components will be instantiated and subsequently reused for the other phase, thus reducing the startup latency of GNN inference which would otherwise be much higher if a pipeline structure was employed for the two phases as in [5].
- **Utilization friendly:** Given the workload allocation scheme as introduced later in Sec. IV-C, all sub-accelerators work on different parts of the feature/weight data in parallel; the hardware utilization and latency can thus be further improved as each sub-accelerator is configured to better fit the corresponding data structure and thus can utilize more parallelism/reuse opportunities.

In particular, the aforementioned template is composed of (1) multiple sub-accelerators which are able to handle both the sparse and dense matrix multiplications, (2) the off-chip memory which holds the data that cannot be stored entirely on chip, and (3) a host CPU to manage the states of different sub-accelerators. Each sub-accelerator has local buffers assigned to the intermediate features (i.e., adjacency matrices), the index for sparse features (assuming a COO format [51]), and the weights and intermediate outputs, respectively. The buffers among sub-accelerators can be configured to be interconnected so that the buffered data can be shared to minimize the costly off-chip memory accesses, as presented in Fig. 3.

**The GNN parsing & sparsity analysis auxiliaries.** Here we briefly describe GNN parsing & sparsity analysis aux-

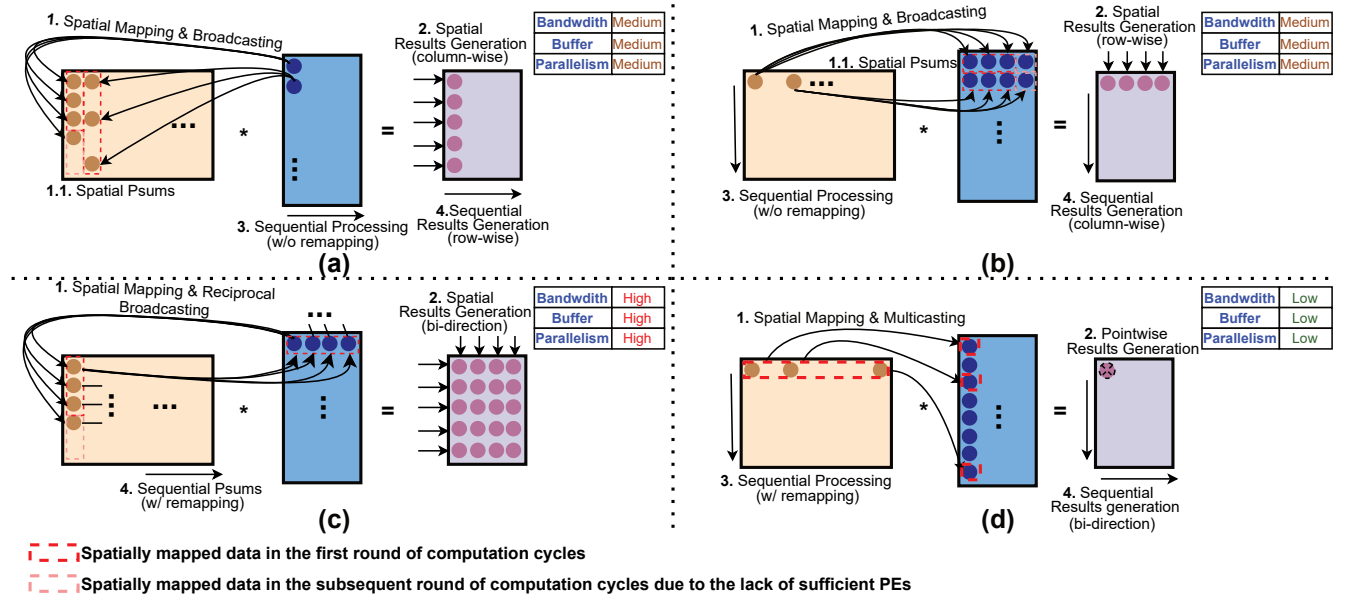


Fig. 4. The choices of kernel modes for each sub-accelerator, where different modes represent different spatial mapping/temporal mapping methods. For better visual clarity, the operation order for each mode is numbered in each sub-figure, and the corresponding properties on off-chip bandwidth consumption, on-chip buffer consumption, and potential parallelism opportunities are summarized on the right corner of each sub-figure.

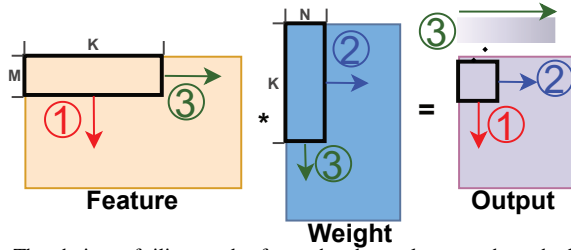


Fig. 5. The choices of tiling modes for each sub-accelerator, where the black box represents the temporal tiles, each arrow color represents the first direction that the tile will move towards, and the color gradient represents the output process from start to complete. The three arrow choices favor weight reuses, feature reuses, and output reuses, respectively. For instance, when tiles first move along the red arrow direction, the tiles of the weights can stay stationary.

iliaries for a given GNN, e.g., from PyG [52]. As shown in Fig. 2, a pre-trained GNN choice is first passed through the blocks of GNN parser and sparsity analyzer before being loaded for G-CoS's automated accelerator generation. Specifically, (1) the GNN parser helps to extract the GNN structure parameters (e.g., the dimensions of the weights and features) and (2) the sparsity analyzer analyzes the sparsity of each adjacency matrix row. They together help the accelerator design space generator produce all the possible choices and balance the workloads for each sub-accelerator.

**Flexible workload allocation.** As different GNN structures can have drastically different sparsity patterns and feature/weight dimensions, G-CoS employs *two flexible workload allocation schemes* to ensure each sub-accelerator's assigned workload better fit its micro-architecture, e.g., the processing element (PE) array's dimensions and tiling sizes, to achieve high hardware utilization and thus efficiency. The main allocation principle is that the assigned workload is proportional to each sub-accelerator's capacity which is characterized by its number of PEs. The two schemes balance the workload with (1) *the number of feature (i.e., adjacency matrix) rows* which will be scaled with the pre-analyzed sparsity of the adjacency matrix and (2) *the number of weight columns*, respectively.

**The sub-accelerator design.** Based on G-CoS's micro-architecture template, the sub-accelerators are auto-generated according to different tiling/kernel modes (as introduced below) to be equipped with different functional components for (1) reflecting different data reuse strategies, (2) favoring different resource trade-offs, and (3) supporting the special operations from various GNN structures, aiming to maximize the hardware efficiency on a wide range of GNNs. Specifically, the sub-accelerators consider:

- **Tiling modes/sizes:** The data per assigned workload may not fit the on-chip memory of each sub-accelerator. As shown in Fig. 5, temporal tiling is then enabled to process the workload temporally. With different tiling modes, a sub-accelerator features more reuses of the features (adjacency matrices), weights, and outputs, respectively. The tiling sizes are defined by K, M, and N (see Fig. 5).
- **Kernel modes:** As illustrated in Fig. 4, with different data mapping and processing patterns for the PE array, each sub-accelerator design would have different off-chip bandwidth consumption, on-chip buffer consumption, and parallelism opportunities.

Moreover, each sub-accelerators is equipped with:

- **Dedicated Buffers** to facilitate local reuse opportunities.
- **Dense/SpMM Hybrid Engine** which supports both dense and sparse matrix multiplication within one unit aided with an configurable PE array as in [49].
- **Element-wise Activation Units** to process the non-linear activation operations.
- **Sampling Units** to schedule the node sampling.

To further increase on-chip reuse opportunities and reduce off-chip accesses, the sub-accelerators can support (1) *weight buffer sharing* which inter-connects all the on-chip weight buffers for weight reuses to reduce off-chip accesses and (2) *buffer re-purposing* where the feature, weight and output buffers are inter-changeable, so no/reduced off-chip accesses

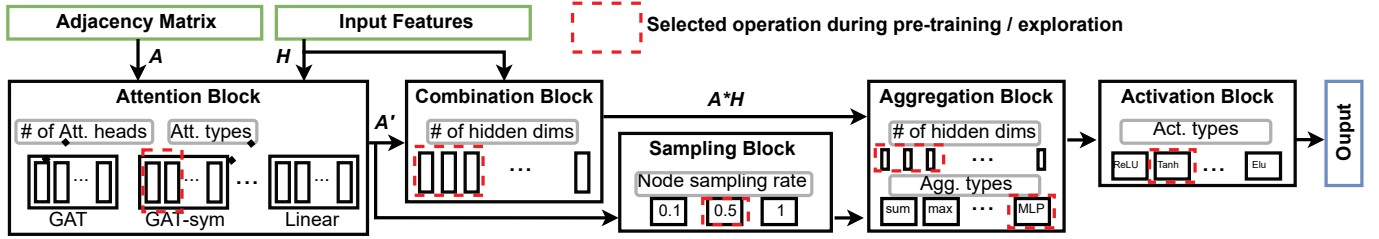


Fig. 6. The GNN supernet of the proposed G-CoS which covers a comprehensive range of GNN options and is compatible with one-shot NAS methods.

TABLE I

THE SEARCHABLE ACCELERATOR PARAMETERS FOR THE PROPOSED G-CoS, WHERE  $n$  DENOTES THE NUMBER OF POSSIBLE TILING SIZES RANGING ABOUT 10~100 DEPENDING ON THE ACCELERATED GNNs.

	Tiling Mode	Kernel Mode	Buffer Re-purposing	WBuf Sharing	Tiling Size
Choice Format	[0,1,2]	[0,1,2,3]	[0,1]	[0,1]	[0,...,n-1]
# of Choices	3	4	2	2	$n$ (~10-100)

are necessary for the intermediate results between the combination and aggregation phases and/or between layers. Considering the controlling complexity and limited on-chip memory size, if either of these two options are enabled, the number of sub-accelerators will be restricted (e.g., 5 in this work).

**The searchable accelerator parameters.** Based on the accelerator template (see Fig. 3), we extract the searchable parameters, of which different combinations lead to different accelerators and form a generic GNN accelerator space to be used by the automated co-search of G-CoS, as summarized in Tab. I. All design choices can be configured differently for each sub-accelerator, except when either *buffer re-purposing* or *weight buffer sharing* is enabled. Otherwise, the tiling and kernel modes are fixed for all sub-accelerators for the ease of controlling and scheduling. The tiling size can range from about 10 to 100 for each sub-accelerator, depending on the given GNN structure. Together, these design choices lead to a hardware design space size of  $1\text{E}+10 \sim 1\text{E}+15$ .

#### D. G-CoS: a generic GNAS search space

**The GNN supernet.** To avoid the retraining cost during co-search, G-CoS incorporates a GNN supernet as its GNN design space which is compatible with the adopted one-shot NAS method and able to produce subnetworks covering a comprehensive range of GNN structures, as shown in Fig. 6. Specifically, the GNN supernet is composed of *five* blocks denoting the attention, combination, sampling, aggregation, and activation blocks. Each block will also have multiple attributes to be determined from a wide range of options as specified in Tab. II. For instance, an attention block may assume a GAT-sym structure and have two heads as in Fig. 6; The combination and aggregation blocks share the same attributes for their hidden dimensions; The attention, sampling, and activation blocks all have a 'skip' option to cover GNNs devoid of these modules. For better generality, each layer of the GNNs assumes this format of supernet, with attribute choices different among layers. For the final/prediction layer, the hidden dimensions and activations are fixed according to the given dataset. Combining all the possible choices in Tab. II, the GNN supernet in G-CoS is able to produce  $\sim 1\text{E}+9$  choices

TABLE II

THE AVAILABLE CHOICES FOR EACH BLOCK ATTRIBUTE IN THE GNN SUPERNET, WITH THE ATTENTION TYPES (ATT. TYPES) FOLLOWING [10].

Att. types	[skip, GCN, GAT, GAT-sym] [COS, Linear, Gene-Linear]
Agg. types	[sum, mean, max, MLP]
Act. types	[Skip, Sigmoid, Tanh, ReLU, Linear] [Softplus, Leaky ReLU, ReLU6, Elu]
# of hidden dims	[4, 8, 16, 32, 64, 128, 256]
# of Att. heads	[1,2,4,6,8,16]
Node sampling rate	[0.1,0.5,1]

for a 2 layer GNN, leading to a joint GNN-accelerator space with more than  $1\text{E}+19$  choices.

**The subnetwork sampling.** The subnetwork is sampled by choosing an option for each attributes of the blocks, e.g., red boxes in Fig. 6. In particular, G-CoS employs uniform random sampling during the pre-training stage, and samples the subnetworks based on the proposed evolutionary algorithm (see Sec. IV-E) during the exploration stage.

#### E. G-CoS: one-shot evolutionary GNN-accelerator co-search

To tackle the aforementioned challenge of excessively large GNN-accelerator joint search space and costly retraining rooted in many NAS methods [10], [35], we propose to employ an one-shot based search approach as inspired by [53], to decouple the supernet pre-training and GNN-accelerator co-search processes, along with an evolutionary algorithm to efficiently navigate through the large joint space to locate optimal GNN-accelerator design pairs for boosting both task accuracy and hardware efficiency. In particular, we only pre-train the GNN supernet once and only inference on the validation set as needed during the exploration stage. To the best of our knowledge, we are the first to study the effectiveness of one-shot NAS within the scope of GNNs.

**The supernet pre-training.** During supernet pre-training, a random subnetwork is uniformly sampled from the GNN supernet by selecting the attribute options from each block and then the subnetwork weights  $\omega$  are updated via back-propagation. As [53] pointed out, uniform sampling can decouple the weights among possible subnetworks and thus provide a better estimate for their individual fully trained accuracy when these subnetworks are inferenced on the validation dataset during the GNN structure exploration.

**The weight sharing.** For more effective pre-training, G-CoS adopts a weight sharing strategy as inspired by [10], [53] during pre-training, such that different subnetworks share certain slices of weights. In particular, the weights for combination and aggregation will be shared and retrieved according to the

---

**Algorithm 1** G-CoS's GNN-accelerator co-search algorithm

---

```
1: Inputs: the target performance  $T$ ; the number of outputs  $N_2$ ; the
   samples pool size  $p_{max}$ , the fitness function  $fit()$ ; the mutation
   function  $mut()$ ; the birth/dying rate  $s$ 
2: Outputs:  $N_2$  number of best found designs  $O_{N_2}$ 
3: [Procedures]:
4:  $P = \{\}$ ;  $fit_{avg} = 0$ 
5: While  $fit_{avg} < T$ 
6:   If  $|P| \leq p_{max}$ 
7:     If  $|P| == 0$ 
8:       Randomly generate  $(s * p_{max})$  new designs  $pnew_{[gnet,hw]}$ 
9:     Else
10:      Find top  $(s * p_{max})$  fit designs  $top_{[gnet,hw]}$ 
11:      Mutate for new designs  $pnew_{[gnet,hw]} = mut(top_{[gnet,hw]})$ 
12:      Evaluate new designs for fitness:  $fit(pnew_{[gnet,hw]})$ 
13:      Add  $\{pnew_{[gnet,hw]}, fit(pnew_{[gnet,hw]})\}$  to  $P$ 
14:    Else
15:      Remove the bottom  $s * p_{max}$  designs from  $P$ 
16:       $fit_{avg} =$  average fitness of top  $N_2$  designs in  $P$ 
17: Return  $N_2$  top fit designs  $O_{N_2}$ 
```

---

chosen # of hidden dimensions when a subnetwork is sampled. For the attention block, only the options belonging to the same attention types share their weights and are retrieved according to the number of attention heads.

**The evolutionary co-search algorithm.** As illustrated in Alg. 1, a specially tailored evolutionary algorithm is developed to efficiently search for best satisfied GNN-accelerator pairs, characterized by  $(gnet, hw)$ . Overall, it operates by constantly generating new designs around the good design options stored in a pool  $P$ , filtering out the inferior designs, and then outputting the top  $N_2$  performing designs. Specifically, the algorithm inputs include: (1) the fitness function  $fit()$ , which evaluates the designs given the GNN-accelerator specs  $(gnet, hw)$ , (2) mutation function  $mut()$  which randomly changes the attributes of good designs to generate new designs, (3) the largest samples pool size  $p_{max}$ , (4) the birth/dying rate which controls how many designs will be generated/filtered out, and (5) performance target  $T$  which determines the terminating conditions and follows the same units as the fitness function. After search, the algorithm outputs the top  $N_2$  performing designs within  $P$ . More input specification choices are provided in Sec.V-A.

## V. EXPERIMENT RESULTS

In this section, we first introduce the experiment setups in Sec. V-A, and then benchmark the proposed G-CoS with SOTA GCN accelerators, GNAS and handcrafted GNNs in Sec. V-B, Sec. V-C and Sec. V-D, respectively.

### A. Experiment Setup

**Baselines and datasets.** For evaluating G-CoS over SOTA GNN accelerators, we consider three baselines: HyGCN [12], AWB-GCN [5], and Deepburning-GL [14], respectively. For evaluating G-CoS over SOTA GNAS, we consider three GNAS baselines: GraphNAS [10], Auto-GNN [34], and Auto-Graph [35]. For benchmarking over SOTA handcrafted GNNs, we consider four baselines: GCN [24], GAT [28], LGCN [30], and GraphSAGE [7], covering the most common GNN variants. Our experiments are conducted on four datasets: three

citation graph datasets (Cora, CiteSeer and Pumbed) [54], and the Reddit post dataset [7]), respectively.

**GNN training setup.** For the GNN training, we follow the same dataset splits as [7], [24], [55]. The GNN supernet is trained using an Adam optimizer [56] with a learning rate of 0.001 for 1000 epochs, L2 regularization, and dropout. After the design space exploration is finished, the final derived models are trained with additional 400 epochs from scratch under the same configurations.

**Evolutionary search algorithm setup.** For the evolutionary search algorithm presented in Alg. 1, we use a set of generic configurations. Specifically, the fitness function is set as the weighted sum of (inverse)latency and task accuracy, such that latency and task accuracy contributes similarly to the fitness score. The mutation function is set to have a 50% chance, i.e., the selected GNN-accelerator design pairs have half of the randomly picked design attributes changed. The birth/dying rate ( $s$ ) is set to 0.2.

**Hardware experiment setup.** To evaluate G-CoS's generated accelerators, we adopt standard the FPGA evaluation and implementation flows in Vivado 2020.2 [57]. For the platform, we picked the Xilinx VCU128 FPGA board [50], which is equipped with 9024 DSPs, 42MB on-chip memory and 460GB/s HBM of-chip memory. For a fair comparison with other baselines, we limit the DSP consumption to be less than 4096 throughout the design. The generated GNN accelerators are clocked at 330MHz and adopt a 16-bit fixed point precision. For the accelerator template, we fix the number of sub-accelerators to 5 (unless otherwise specified). Note that the design principles of G-CoS introduced in Sec. IV-C is platform agnostic, i.e., G-CoS can be flexibly extended to other platforms such as ASIC. During the design space exploration, to quickly go over numerous design options, we design and implement an in-house performance simulator to measure the execution time (i.e., the number of cycles).

### B. G-CoS over SOTA GNN accelerators

In this set of experiments, we compare G-CoS with existing SOTA GNN accelerators: HyGCN [12], AWB-GCN [5], and Deepburning-GL [14], in terms of latency and bandwidth consumption. For a fair comparison, we fix the GNN structures and datasets in G-CoS to be the same as the baselines and only search for the accelerator parameters. Since most of them do not provide absolute performance values while reporting the relative speedups over PyG-CPU on an Intel Xeon E5-2680 v3 CPU instead. We also measure and verify the latency on the same CPU and calculate the FPGA speedups over it, so that PyG-CPU is a common baseline for all methods, and then we can analyze the relative improvements as elaborated below:

(1) **Speedup.** As shown in Fig. 8, G-CoS achieves an average of  $5.52\times$ ,  $1.92\times$ ,  $35.98\times$ , and  $21.54\times$  speedups over HyGCN, AWB-GCN, Deepburning-GL-KCU1500, and Deepburning-GL-Alveo U50, respectively. G-CoS's much reduced latency is mostly attributed to its enabling better hardware(DSP) utilization as the multi-sub-accelerator scheme of G-CoS can better cover GNNs' high irregularity and the wide



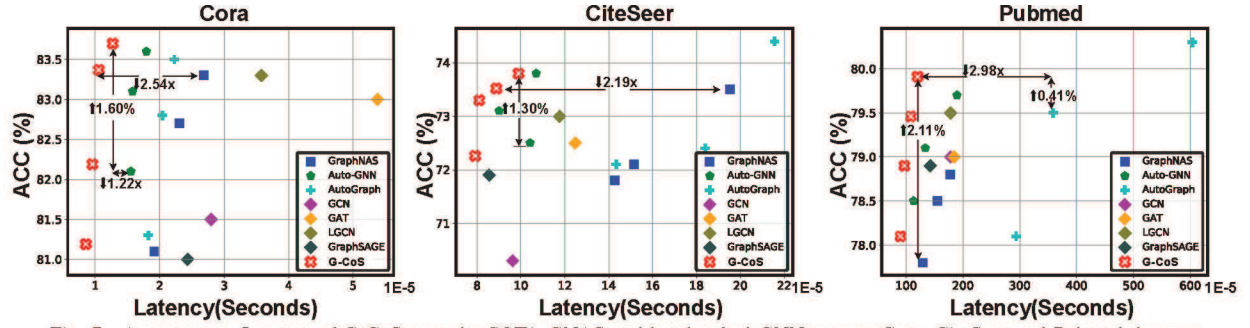


Fig. 7. Accuracy vs. Latency of G-CoS over the SOTA GNAS and handcrafted GNNs across Cora, CiteSeer and Pubmed datasets.

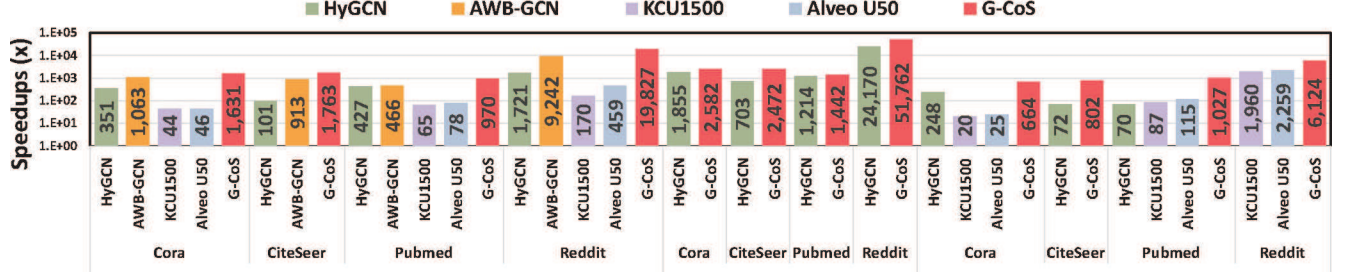


Fig. 8. The normalized inference speedups (w.r.t. PyG-CPU) achieved by G-CoS over the four SOTA baseline platforms on three GNN models and four representative graph datasets, where KCU1500 and Alveo U50 are two implementation platforms for Deepburning-GL [14]

range of searchable accelerator parameters make G-CoS's searched accelerators matching different variation in GNNs. (2) Off-chip memory Bandwidth Consumption. G-CoS only requires an average of 50% off-chip memory bandwidth as compared to HyGCN. The high bandwidth of HyGCN is due to its high-degree parallelism while G-CoS's versatile kernel mode as illustrated in Sec. IV-C alleviates such an off-chip bandwidth pressure.

### C. G-CoS over SOTA GNAS.

In this set of experiments, we benchmark G-CoS with the SOTA GNAS works: GraphNAS [10], Auto-GNN [34], and AutoGraph [35] based on the metrics of task accuracy and latency. For G-CoS, we co-optimize both the GNN and accelerator design parameters as introduced in Sec. IV-C and Sec. IV-D, respectively. For a fair comparison, we also accelerate the baselines' generated GNNs under the same hardware platform and optimize the corresponding accelerator parameters. To demonstrate the tradeoff between the hardware efficiency and task accuracy, we restricted the baseline generated GNNs with different flops and then select the designs with the lowest latency; for G-CoS, we simply decrease/increase the weighting coefficient of the latency in the search metrics to achieve flexible tradeoffs. The results are presented in Fig. 7: G-CoS consistently maintains a better performance frontier, i.e., a higher accuracy and lower latency. In particular, G-CoS achieves a 1.60%, 1.3% and 2.11% increase in task accuracy with a similar or lower latency, and a 2.54 $\times$ , 2.15 $\times$  and 2.98 $\times$  latency reduction with a similar or higher accuracy, as compared with the SOTA GNAS works on the Cora, CiteSeer and Pubmed datasets, respectively. Although, considering flops during the search can offer some guidance for more hardware-friendly GNNs, it can not fully capture the compatibility between the searched GNNs and the specific hardware platform, resulting the searched GNNs which satisfy the flops requirement but might be hard to accelerate. Thus,

co-optimizing the GNN-accelerator pairs can excel both in terms of accuracy and hardware efficiency against traditional GNAS works by offering better guidance and fully customized architecture to every single searched network. The entire process takes as low as 4 GPU hours depending on the dataset.

### D. G-CoS over SOTA handcrafted GNNs.

We also compare the performance of the proposed G-CoS against the SOTA handcrafted GNNs: GCN [24], GAT [28], LGCN [30], and GraphSAGE [7]. For G-CoS, we co-optimize the GNN-accelerator design pairs. For the baselines GNNs, we optimize their accelerator parameters for fair comparison. As shown in Fig. 7, G-CoS's generated GNN-accelerator design pairs consistently achieve better accuracy and lower latency at the same time. Specifically, the G-CoS generated designs achieve up to 2.7%, 3.22% and 1.01% increase in accuracy while having 1.91 $\times$ , 1.08 $\times$  and 1.19 $\times$  reduction in latency.

## VI. CONCLUSION

We propose G-CoS, a GNN and accelerator co-search framework to automatically search for matched GNN structures and accelerators to maximize both task accuracy and acceleration efficiency. To the best of our knowledge, G-CoS is the first co-search framework for GNNs and their accelerators. Extensive experiments show that the GNNs and accelerators generated by G-CoS consistently outperform SOTA GNNs and GNN accelerators, while only requiring a few hours for the end-to-end generation of the matched GNNs and their accelerators. We believe that our G-CoS has made an important heuristic step towards boosted GNN acceleration efficiency and fast development of efficient GNN solutions.

## ACKNOWLEDGMENT

This work was supported in part by the National Institutes of Health under Award R01HL144683, National Science Foundation under CAREER-2048183.



## REFERENCES

- [1] T. N. Kipf and M. Welling, "Semi-supervised classification with graph convolutional networks," *arXiv preprint arXiv:1609.02907*, 2016.
- [2] K. Xu, W. Hu, J. Leskovec, and S. Jegelka, "How powerful are graph neural networks?" *arXiv preprint arXiv:1810.00826*, 2018.
- [3] R. Ying, R. He, K. Chen, P. Eksombatchai, W. L. Hamilton, and J. Leskovec, "Graph convolutional neural networks for web-scale recommender systems," in *Proceedings of the 24th ACM SIGKDD International Conference on Knowledge Discovery & Data Mining*, 2018, pp. 974–983.
- [4] S. A. Tailor, J. Fernandez-Marques, and N. D. Lane, "Degree-quant: Quantization-aware training for graph neural networks," in *International Conference on Learning Representations*, 2021. [Online]. Available: <https://openreview.net/forum?id=NSBrFgJAHg>
- [5] T. Geng *et al.*, "Awb-gcn: A graph convolutional network accelerator with runtime workload rebalancing," in *MICRO*, 2020.
- [6] A. A. Awan, H. Subramoni, and D. K. Panda, "An in-depth performance characterization of cpu- and gpu-based dnn training on modern architectures," in *Proceedings of the Machine Learning on HPC Environments*, ser. MLHPC'17. Association for Computing Machinery, 2017.
- [7] W. Hamilton *et al.*, "Inductive representation learning on large graphs," in *NeurIPS*, 2017.
- [8] J. Li *et al.*, "Sgcn: A graph sparsifier based on graph convolutional networks," in *Pacific-Asia Conference on Knowledge Discovery and Data Mining*, 2020.
- [9] C. Zheng, B. Zong, W. Cheng, D. Song, J. Ni, W. Yu, H. Chen, and W. Wang, "Robust graph representation learning via neural sparsification," in *International Conference on Machine Learning*. PMLR, 2020, pp. 11 458–11 468.
- [10] Y. Gao *et al.*, "Graphnas: Graph neural architecture search with reinforcement learning," *arXiv preprint arXiv:1904.09981*, 2019.
- [11] J. L. Jiaxuan You, Zhitao Ying, "Design space for graph neural networks," in *Thirty-fourth Conference on Neural Information Processing Systems*, 2020.
- [12] M. Yan *et al.*, "Hygen: A gen accelerator with hybrid architecture," in *HPCA*, 2020.
- [13] R. Garg *et al.*, "A taxonomy for classification and comparison of dataflows for gnn accelerators," *arXiv preprint arXiv:2103.07977*, 2021.
- [14] S. Liang *et al.*, "Deepburning-gl: an automated framework for generating graph neural network accelerators," in *ICCAD*, 2020.
- [15] Y. Zhang *et al.*, "DIAN: Differentiable accelerator-network co-search towards maximal dnn efficiency," in *2021 IEEE/ACM International Symposium on Low Power Electronics and Design (ISLPED)*, 2021, pp. 1–6.
- [16] C. Hao *et al.*, "Fpga/dnn co-design: An efficient design methodology for iot intelligence on the edge," in *DAC*, 2019.
- [17] M. S. Abdelfattah, Ł. Dudziak, T. Chau, R. Lee, H. Kim, and N. D. Lane, "Best of both worlds: Autolml codegen of a cnn and its hardware accelerator," *arXiv preprint arXiv:2002.05022*, 2020.
- [18] Y. Fu *et al.*, "Auto-NBA: Efficient and effective search over the joint space of networks, bitwidths, and accelerators," in *The 38th International Conference on Machine Learning (ICML 2021)*, 2021.
- [19] Y. Zhang *et al.*, "RT-RCG: Neural network and accelerator search towards effective and real-time ECG reconstruction from intracardiac electrograms," in *ACM Journal on Emerging Technologies in Computing Systems*, 2021.
- [20] W. Jiang *et al.*, "Hardware/software co-exploration of neural architectures," *arXiv preprint arXiv:1907.04650*, 2019.
- [21] Y. Li, C. Hao, X. Zhang, X. Liu, Y. Chen, J. Xiong, W.-m. Hwu, and D. Chen, "Edd: Efficient differentiable dnn architecture and implementation co-search for embedded ai solutions," *arXiv preprint arXiv:2005.02563*, 2020.
- [22] M. Zhang *et al.*, "An end-to-end deep learning architecture for graph classification," in *AAAI*, 2018.
- [23] M. a. Gori, "A new model for learning in graph domains," in *IJCNN*, 2005.
- [24] T. N. Kipf and M. Welling, "Semi-supervised classification with graph convolutional networks," in *ICLR*, 2017.
- [25] W. Peng *et al.*, "Learning graph convolutional network for skeleton-based human action recognition by neural searching," in *AAAI*, 2020.
- [26] Z. Wu, S. Pan, F. Chen, G. Long, C. Zhang, and S. Y. Philip, "A comprehensive survey on graph neural networks," *IEEE transactions on neural networks and learning systems*, 2020.
- [27] M. Simonovsky and N. Komodakis, "Dynamic edge-conditioned filters in convolutional neural networks on graphs," in *CVPR*, 2017.
- [28] P. Veličković *et al.*, "Graph attention networks," in *ICLR*, 2018.
- [29] H. Zeng *et al.*, "Accurate, efficient and scalable graph embedding," in *IPDPS*, 2019.
- [30] H. Gao *et al.*, "Large-scale learnable graph convolutional networks," in *KDD*, 2018.
- [31] B. Zoph and Q. V. Le, "Neural architecture search with reinforcement learning," *arXiv preprint arXiv:1611.01578*, 2016.
- [32] H. Pham *et al.*, "Efficient neural architecture search via parameters sharing," in *ICML*, 2018.
- [33] I. Bello *et al.*, "Neural optimizer search with reinforcement learning," in *ICML*, 2017.
- [34] K. Zhou *et al.*, "Auto-gnn: Neural architecture search of graph neural networks," *arXiv preprint arXiv:1909.03184*, 2019.
- [35] G. Kyriakides and K. Margaritis, "Evolving graph convolutional networks for neural architecture search," *Neural Computing and Applications*, 2021.
- [36] J. You *et al.*, "Design space for graph neural networks," in *NeurIPS*, 2020.
- [37] M. Tan, B. Chen, R. Pang, V. Vasudevan, M. Sandler, A. Howard, and Q. V. Le, "Mnasnet: Platform-aware neural architecture search for mobile," in *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, 2019, pp. 2820–2828.
- [38] A. Howard, M. Sandler, G. Chu, L.-C. Chen, B. Chen, M. Tan, W. Wang, Y. Zhu, R. Pang, V. Vasudevan *et al.*, "Searching for mobilenetv3," in *ICCV*, 2019.
- [39] M. Tan and Q. V. Le, "Efficientnet: Rethinking model scaling for convolutional neural networks," *arXiv preprint arXiv:1905.11946*, 2019.
- [40] H. Liu, K. Simonyan, and Y. Yang, "Darts: Differentiable architecture search," *arXiv preprint arXiv:1806.09055*, 2018.
- [41] B. Wu, X. Dai, P. Zhang, Y. Wang, F. Sun, Y. Wu, Y. Tian, P. Vajda, Y. Jia, and K. Keutzer, "Fbnet: Hardware-aware efficient convnet design via differentiable neural architecture search," in *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, 2019, pp. 10 734–10 742.
- [42] A. Wan, X. Dai, P. Zhang, Z. He, Y. Tian, S. Xie, B. Wu, M. Yu, T. Xu, K. Chen *et al.*, "Fbnetv2: Differentiable neural architecture search for spatial and channel dimensions," in *CVPR*, 2020.
- [43] X. Jin *et al.*, "Rc-darts: Resource constrained differentiable architecture search," *arXiv preprint arXiv:1912.12814*, 2019.
- [44] A. Auten *et al.*, "Hardware acceleration of graph neural networks," in *DAC*, 2020.
- [45] S. Liang *et al.*, "Engn: A high-throughput and energy-efficient accelerator for large graph neural networks," *IEEE Transactions on Computers*, 2020.
- [46] K. Kinningham *et al.*, "Grip: A graph neural network accelerator architecture," *arXiv preprint arXiv:2007.13828*, 2020.
- [47] Y. Zhao, X. Chen, Y. Wang, C. Li, H. You, Y. Fu, Y. Xie, Z. Wang, and Y. Lin, "Smartexchange: Trading higher-cost memory storage/access for lower-cost computation," 2020.
- [48] K. Xu *et al.*, "How powerful are graph neural networks?" in *ICLR*, 2019.
- [49] F. Shi *et al.*, "Versagnn: a versatile accelerator for graph neural networks," *arXiv preprint arXiv:2105.01280*, 2021.
- [50] Xilinx Inc., "Virtex ultrascale+ hbm vcu128 fpga evaluation kit," <https://www.xilinx.com/products/boards-and-kits/vcu128.html>, (Accessed on 09/30/2020).
- [51] "Sparse matrix," [https://en.wikipedia.org/wiki/Sparse\\_matrix](https://en.wikipedia.org/wiki/Sparse_matrix), (Accessed on 05/28/2021).
- [52] M. Fey and J. E. Lenssen, "Fast graph representation learning with PyTorch Geometric," in *ICLR Workshop on Representation Learning on Graphs and Manifolds*, 2019.
- [53] Z. Guo *et al.*, "Single path one-shot neural architecture search with uniform sampling," in *ECCV*, 2020.
- [54] P. Sen *et al.*, "Collective classification in network data," *AI magazine*, 2008.
- [55] W. Hu *et al.*, "Open graph benchmark: Datasets for machine learning on graphs," *arXiv preprint arXiv:2005.00687*, 2020.
- [56] D. P. Kingma and J. Ba, "Adam: A method for stochastic optimization," *arXiv preprint arXiv:1412.6980*, 2014.
- [57] Xilinx Inc., "Vivado Design Suite - HLx Editions," <https://www.xilinx.com/products/design-tools/vivado.html>, accessed 2019-09-16.