



MaGNAS: A Mapping-Aware Graph Neural Architecture Search Framework for Heterogeneous MPSoC Deployment

MOHANAD ODEMA, University of California Irvine, USA

HALIMA BOUZIDI, HAMZA OUARNOUGHI, and SMAIL NIAR, Université Polytechnique Hauts-de-France, France

MOHAMMAD ABDULLAH AL FARUQUE, University of California Irvine, USA

Graph Neural Networks (GNNs) are becoming increasingly popular for vision-based applications due to their intrinsic capacity in modeling structural and contextual relations between various parts of an image frame. On another front, the rising popularity of deep vision-based applications at the edge has been facilitated by the recent advancements in heterogeneous multi-processor Systems on Chips (MPSoCs) that enable inference under real-time, stringent execution requirements. By extension, GNNs employed for vision-based applications must adhere to the same execution requirements. Yet contrary to typical deep neural networks, the irregular flow of graph learning operations poses a challenge to running GNNs on such heterogeneous MPSoC platforms. In this paper, we propose a novel unified *design-mapping* approach for efficient processing of vision GNN workloads on heterogeneous MPSoC platforms. Particularly, we develop MaGNAS, a mapping-aware Graph Neural Architecture Search framework. MaGNAS proposes a GNN architectural design space coupled with prospective mapping options on a heterogeneous SoC to identify model architectures that maximize on-device resource efficiency. To achieve this, MaGNAS employs a two-tier evolutionary search to identify optimal GNNs and *mapping* pairings that yield the best performance trade-offs. Through designing a supernet derived from the recent Vision GNN (ViG) architecture, we conducted experiments on four (04) state-of-the-art vision datasets using both (i) a real hardware SoC platform (NVIDIA Xavier AGX) and (ii) a performance/cost model simulator for DNN accelerators. Our experimental results demonstrate that MaGNAS is able to provide $1.57\times$ latency speedup and is $3.38\times$ more energy-efficient for several vision datasets executed on the Xavier MPSoC vs. the GPU-only deployment while sustaining an average **0.11%** accuracy reduction from the baseline.

CCS Concepts: • **Computing methodologies** → **Distributed computing methodologies**; **Neural networks**; • **Computer systems organization** → *Embedded and cyber-physical systems*;

Additional Key Words and Phrases: Graph neural networks, MPSoCs, HW-SW codesign, edge computing

M. Odema and H. Bouzidi contributed equally to this research.

This article appears as part of the ESWEEK-TECS special issue and was presented in the International Conference on Compilers, Architectures, and Synthesis for Embedded Systems (CASES), 2023.

Authors' addresses: M. Odema and M. A. Al Faruque, University of California Irvine, Irvine, USA; emails: {modema, alfaruqu}@uci.edu; H. Bouzidi, H. Ouarnoughi, and S. Niar, Université Polytechnique Hauts-de-France, Valenciennes, France; emails: {Halima.Bouzidi, Hamza.Ouarnoughi, Smail.Niar}@uphf.fr.

Permission to make digital or hard copies of part or all of this work for personal or classroom use is granted without fee provided that copies are not made or distributed for profit or commercial advantage and that copies bear this notice and the full citation on the first page. Copyrights for third-party components of this work must be honored. For all other uses, contact the owner/author(s).

© 2023 Copyright held by the owner/author(s).

1539-9087/2023/09-ART108 \$15.00

<https://doi.org/10.1145/3609386>

ACM Reference format:

Mohanad Odema, Halima Bouzidi, Hamza Ouarnoughi, Smail Niar, and Mohammad Abdullah Al Faruque. 2023. MaGNAS: A Mapping-Aware Graph Neural Architecture Search Framework for Heterogeneous MPSoC Deployment. *ACM Trans. Embedd. Comput. Syst.* 22, 5s, Article 108 (September 2023), 26 pages. <https://doi.org/10.1145/3609386>

1 INTRODUCTION

Due to their inherent capacity in learning meaningful feature representations from non-Euclidean graph-structured data, the employment of Graph Neural Networks (GNNs) has extended beyond typical graph learning applications, e.g., molecular inference and social networks [32], to encompass the field of computer vision. By transforming an image structured as a regular grid of pixels into a graph, irregular and complex objects can be better captured by the more flexible graph-level features generated throughout the model architecture. As such, recent works employing GNNs to operate on this generalized form of image data have demonstrated remarkable successes across a variety of visual tasks, e.g., object detection and image classification [17, 31, 37, 38]. In fact, the application of GNNs has been further studied for more nuanced visual-based tasks in critical application settings, such as collision prediction in self-driving vehicles [25, 43].

On a separate note, recent advances have seen a proliferation in multi-processor System-on-Chips (MPSoCs) architectures that can balance the low-latency and energy efficiency requirements of compute-intensive workloads. For instance, commercial SoC platforms, such as the Nvidia Xavier [1] and Tesla FSD [30], have successfully integrated a variety of proven hardware computing units (CUs) and industrial IPs on a single chip to achieve said purpose. Other platforms, such as Xilinx Versal [12], enable even more flexibility in SoC solution development by supporting customized hardware design choices. Through such advanced platforms, deep learning-based vision modules can be run effectively in an edge computing setting to meet stringent application requirements such as object detection for autonomous driving [24]. By extension, any consideration for applying GNNs in these vision modules under the embedded deployment setting must ensure that the execution constraints are still satisfied. However, this objective is challenging, considering the discrepancy between the GNN workloads and the underlying hardware in the SoC. That is, contrary to the dense, regular workloads of typical DNNs, GNNs are characterized by an *irregular, multiphase sparse-dense* computational flow [15]. Particularly, this irregularity emanates from the repeated sequence of *Aggregation* and *Combination* phases. The former employs a message-passing algorithm for feature exchange between graph vertices, exhibiting sparse kernels with random memory access patterns. The latter constitutes typical multi-layer perceptron (MLP) layer(s) for feature transformation, exhibiting dense kernels and regular access patterns. As such, the complication arises as neither the architecture of typical CUs (e.g., GPU) nor that of conventional accelerators (e.g., DLA) is designed to efficiently support this unique execution sequence.

Naturally, considerable research works have dedicated efforts to design customized GNN accelerator architectures that can support the *multi-phased* computational flow [3, 7, 19, 29, 36, 39]. Generally, the approach entailed a *hybrid* architecture comprising specialized computing engines to accelerate each of the two phases separately. Unfortunately, these designs are not flexible enough to be consolidated into standard MPSoCs. On the one hand, this is attributed to the fact that GNNs belong to a relatively nascent, rapidly-evolving field in which customized accelerator architectures may not support running newer generations of graph learning operations and models. On the other hand, physical restrictions and low-power requirements of critical embedded computing platforms at the edge *restrict* the integration of specialized hardware CUs onto the SoC to the components

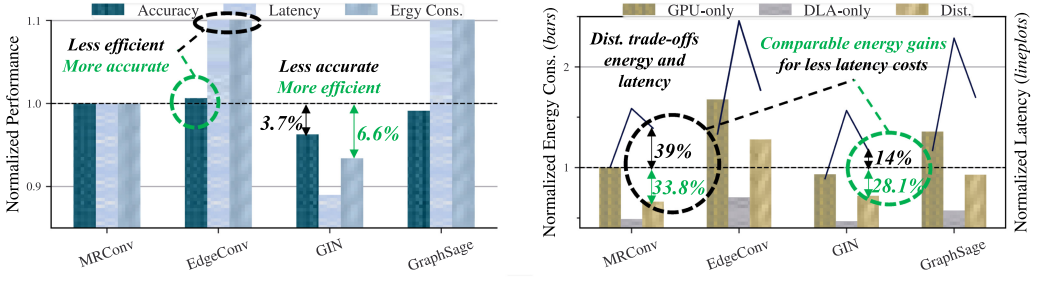


Fig. 1. Comparing ViG model variants [17] with different graph learning operators when trained on the Oxford-Flowers dataset and deployed onto the NVIDIA Jetson AGX Xavier SoC. All values are normalized by the baseline performance evaluations incurred by the original ViG with MRConv layers when fully deployed onto the GPU only. The *left* figure shows how performance characteristics differ from one variant to the other regarding accuracy, latency, and energy consumption. The *right* figure illustrates how distributed mapping strategies across the GPU and DLA can yield different latency-energy trade-offs.

that best serve the desired target applications – as in how DLAs are integrated in the AGX Xavier SoC as they support a broad class of applications which employ typical DNN workloads.

As GNNs continue to become increasingly popular, the challenges of their deployment onto embedded platforms are due to be seen in a new light. In addition to implementing customized accelerator architectures, another research direction is to investigate what optimization opportunities exist – on both the hardware and algorithmic levels – to alleviate the deficiencies of GNNs’ computational flow when deployed on conventional CUs. Researchers in [15] have assumed this perspective by characterizing the design space of *dataflow* choices for running GNNs on conventional re-configurable spatial accelerators, where they studied the costs and benefits of adopting various dataflows for GNNs. In that same spirit, we also believe there are ample optimization opportunities through characterizing the combined design space of *SoC mapping options* and *GNN architectural parameters* together. In the context of GNNs for vision applications, two considerations motivate this hypothesis: (i) Heterogeneous MPSoCs naturally offer *pipelining parallelism* opportunities, presenting options to run GNN kernels of diverse characteristics on different CUs to potentially yield better performance benefits. (ii) the recently proposed VisionGNN (ViG) architecture [17] offers to transform an image frame into a graph by dividing it into equally-sized patches and constructing a graph out of them to be processed by the model. As will be detailed later, the key advantage of this scheme is that it enables leveraging graph-level features while maintaining a consistent, dense structure for any graphed image throughout the GNN model, which is more amenable to CUs than sparse graphs of *inconstant* dimensions.

1.1 Motivational Example

In Figure 1, we showcase the potential performance trade-offs as offered by the *architectural* and *mapping* optimization spaces for a vision GNN model when deployed onto a heterogeneous SoC. In this example, the backbone GNN architecture is the ViG-S [17], the target platform is the NVIDIA Xavier AGX SoC, and the models are trained on the Oxford-Flowers image dataset. Given how the ViG belongs to the Graph Convolutional Network (GCN) class of GNNs, we construct three (03) additional variants of the baseline ViG with different GCN operators. Specifically, the original ViG architecture employs the Max-Relative Graph Conv (MRConv) graph operation throughout the entirety of its model, whereas the variants employ other GCN layer types, namely EdgeConv, GIN, and GraphSage. After training the ViG variants, we characterize their accuracy, latency, and

energy consumption scores relative to the original MRConv ViG variant when deployed onto the NVIDIA platform. In the *left* Figure, we can observe some performance trade-offs from varying this *singular* GNN architectural setting, i.e., the GCN layer operator. For instance, the EdgeConv ViG variant can achieve slightly higher accuracy (0.69% more) than the MRConv one at the expense of a considerable increase in latency and energy consumption. Contrarily, the GIN operation is 6.6% more energy-efficient than MRConv at the expense of a 3.7% decrease in accuracy. Though there is no clear dominance for one variant over the other, this analysis sheds light on the potential performance trade-off gains from optimizing the architectural design parameters. These gains can be further compounded when considered alongside feasible deployment options. In these first experiments, only the GPU component of the SoC was used as the target deployment hardware.

In the *right* Figure, we showcase how additional performance trade-offs are attained considering the various deployment options for the ViG variants on the SoC. In this example, the considered options are *standalone* deployment on either the GPU or DLA components or *distributed* deployment across the two. We remark that the distributed deployment options follow the mapping strategies for GNN processing workloads provided by our optimization engine, detailed in a later Section. From the Figure, the straightforward observation is that for every ViG architecture, *standalone* GPU deployment is the option with the fastest execution speeds, *standalone* DLA deployment is the most energy-efficient alternative, and the distributed option compromises between the two. However, a more interesting perspective on mapping optimizations can be taken when considered part of a broader design problem. That is, combining both the *architectural* and *mapping* optimizations to achieve better performance trade-offs compared to performing optimizations for each design space in isolation. For instance, assume a designer's primary objective is to improve the ViG's energy efficiency while incurring minimal execution slowdown. From a pure resource efficiency perspective, a *distributed* mapping strategy for the GIN *architectural* variant can be more beneficial than directly distributing the original MRConv ViG workloads since the former achieves comparable energy efficiency gains to those of the latter (28.1% to 33.8%) at the expense of reduced latency costs (14% to 39%). Still, the caveat remains that the GIN variant is less accurate than the original ViG, and the question becomes *how can we better characterize this combined architecture-mapping design space to attain better performance trade-offs for vision GNNs given the target task and SoC platform*.

1.2 Novel Contributions

In light of the above challenges, we list the key novel contributions of this paper:

- We study how vision GNNs can leverage distributed deployment across multiple CUs for performance efficiency when deployed onto a heterogeneous SoC.
- We present **MaGNAS**, a Mapping-aware Graph Neural Architecture Search Framework for *co-optimizing* the design of vision GNN (ViG) architectures and their SoC mappings.
- MaGNAS first contributes a self-contained framework for designing ViG supernet to characterize their search space of GNN-based architectural design choices.
- To specify the mapping problem, we derive a system model that characterizes the distributed deployment of GNNs onto heterogeneous SoCs and the incurred performance overheads.
- To identify optimal *ViG architecture-mapping* pairs, MaGNAS solves a bilevel optimization problem via a two-tier evolutionary search algorithm of two optimization engines: an *outer* engine to optimize GNN architectural design choices; an *inner* engine to identify optimal mapping strategies for ViG workloads onto heterogeneous CUs.
- We conduct extensive experiments, in-depth analysis, and ablation studies on MaGNAS using a real MPSoC platform and hardware simulator on four (04) state-of-the-art vision

datasets. Our findings have demonstrated the superiority of MaGNAS in designing and mapping ViG architectures onto heterogeneous CUs and its effective scaling capabilities on increasing levels of problem complexity. On the Nvidia Xavier SoC, MaGNAS provided on average $1.57\times$ latency speedup and $3.38\times$ more energy gains than the GPU-only deployment while sustaining an average 0.11% accuracy drop from the baseline.

2 A PRIMER ON VISION GRAPH NEURAL NETWORK (ViG)

We briefly describe the main constituents of the ViG architecture [17], which pioneered a generic approach for graph-based image processing through modeling raw input images as graph structures.

Graphing Image Data Structures. The ViG operates on images modeled as graphs of patches. A $W \times H \times C$ image is first partitioned into N patches of dimensions $W' \times H' \times C'$. Each patch's dimensions can be viewed as a single feature vector $x_i \in \mathbb{R}^D$ where $D = W' \times H' \times C'$. To construct the graph, a node v_i is assigned to each patch, forming an unordered set of N nodes $\mathcal{V} = \{v_1, v_2, \dots, v_N\}$ associated with the corresponding set of feature vectors $X = \{x_1, x_2, \dots, x_N\}$, where x_i can be called the *feature embedding* of vertex v_i . To build graph edges, K edges are constructed for each v_i based on the K nearest vertices in its neighborhood $\mathcal{N}(\mathcal{V})$, that is, for every $v_j \in \mathcal{N}(\mathcal{V})$, an edge e_{ji} is constructed from v_j to v_i . Finally, the full graph structure of the image is given by $\mathcal{G}(\mathcal{V}, \mathcal{E})$, which can be inputted into the ViG model for processing.

Graph Processing Layer. Describing a graph through its features, $\mathcal{G} = G(X)$ s.t. $X \in \mathbb{R}^{N \times D}$, a typical GCN layer operation on $\mathcal{G}$ can be represented by the following abstract formula:

$$\mathcal{G}' = \text{Combine}(\text{Aggregate}(\mathcal{G}, W_{agg}), W_{comb}) \quad (1)$$

where $\mathcal{G}$ is processed through an *aggregation* and a *combination* stages of the GCN layer. W_{agg} and W_{comb} resemble the respective learnable weights of each stage. The *aggregation* stage employs a feature exchange procedure in which every node v_i receives features $x_j \in \mathcal{N}(x_i)$ s.t. $i \neq j$ from its neighboring nodes and aggregates them to provide x'_i . The *combination* stage involves further treatment of features x'_i (as through an MLP layer) to obtain refined representation x''_i . We remark that for each of the two stages, a variety of operations can be employed (e.g., aggregation through sum, max-relative, mean), which correspond to the variety of GCN layer types existing in the literature (e.g., GraphSage, GIN, etc.). Lastly, The resulting output feature set from both stages, X' , is used to construct the output graph $\mathcal{G}' = G(X')$.

Grapher and FFN Modules. To enrich feature representation, graph processing layers can be interleaved with typical DNN layers in a GNN model. As such, the standard ViG architecture comprises a stack of two basic building blocks: *Grapher* and *Feed Forward Network (FFN)* given by:

$$L^{Grapher} = l^{post} \circ l^{comb} \circ l^{agg} \circ l^{pre}, \quad L^{FFN} = l^{fc_2} \circ l^{fc_1} \quad (2)$$

The *Grapher* comprises at its core the GCN layer with its aggregation, l^{agg} , and combination, l^{comb} , operations, injected between two linear layers, namely *pre-processing*, (l^{pre}), and *post-processing*, (l^{post}), layers, to promote feature diversity. The *FFN* block constitutes two fully connected layers that further elevate feature capacity, l^{fc_1} and l^{fc_2} . For every GCN or fully-connected layer in either module, non-linear activation and batch normalization operations are applied. From here, every *Grapher* can be followed by an optional *FFN* to form the ViG block, and the sequence of ViG blocks form the ViG backbone architecture.

3 SYSTEM MODEL AND PROBLEM FORMULATION

In this section, we model the mapping problem of GNN kernels onto heterogeneous SoC CUs. Then, we derive a formulation for the global design-mapping bi-optimization objective.

3.1 System Model for Mapping GNNs onto Heterogeneous SoCs

3.1.1 GNN Workload Characterization. Let a standard GNN model architecture, α , be formally described as a sequence of n computing blocks as follows:

$$\alpha = L_n \circ L_{n-1} \circ \dots \circ L_1, \text{ s.t. } L_i \neq L_{i-1}, L_i \in \{L^{FFN}, L^{Grapher}\}, L^{FFN} \in \{L^{FFN}, \phi\} \forall 1 \leq i \leq n \quad (3)$$

where each GNN computing block L_i can either be the *Grapher* or *FFN* blocks as defined in the previous section, denoted by $L^{Grapher}$ and L^{FFN} , respectively. The condition ensures that each $L^{Grapher}$ block can be succeeded by an optional L^{FFN} block.

Let X_j be the input graph-level features for block $L_j \in \alpha$. Then, the output feature embedding vector, X_{j+1} , can be obtained as:

$$X_{j+1} = L_j(X_j) \text{ s.t. } x_k^j \in \mathbb{R}^{D'} \forall x_k^j \in X_j \quad (4)$$

where the condition ensures that feature embedding dimensions remain consistent throughout each computing block within the GNN. That is the feature embedding for x_k^j (the k^{th} node within the graph representation at the j^{th} block) retains the same D' dimensions before and after being processed through block L_j . This consistency in the feature embedding dimensions is typical of GNNs as it preserves the integrity of graph operations with regards to feature aggregation from farther nodes across multiple consecutive layers and facilitates supporting residual and dense connections [40]. Note that D' can either be equivalent to D or a downsampled version of it as some architectures (e.g., Pyramid in [17]) can include additional downsampling layers in-between stacks of computing blocks to promote abstract feature learning.

Let $\mathbb{CU} = \{\mathcal{CU}_1, \mathcal{CU}_2, \dots, \mathcal{CU}_M\}$ be the set of available computing units within a heterogeneous MPSoC with varying degrees of support for DNN and graph operations. Considering a *blockwise* granularity, we can define a mapping vector, m , to characterize the workload distribution for each GNN computational block as follows:

$$m = [\pi_1, \pi_2, \dots, \pi_n], \text{ s.t. } \pi_i \in \mathbb{CU} \forall 1 \leq i \leq n \mid \text{support}(\pi_i, L_i) == \text{True} \quad (5)$$

where each entry π_i in $\mathbb{M}$ describes the mapping assignment of L_i onto a computing unit $\mathcal{CU}_m \in \mathbb{CU}$ as long as this corresponding $\mathcal{CU}_m$ hardware supports running L_i .

3.1.2 Performance Modelling. For a mapping strategy m , the total latency and energy consumption overheads, T_{total} and E_{total} , experienced by a GNN model when deployed in a distributed, pipelined fashion can be modeled as the sum of the overheads incurred by its individual blocks:

$$T_{total}(m) = \sum_{i=1}^n T_i(m), \text{ s.t. } T_i(m) = \tau_i^{comp} + \mathbb{I}[\pi_{i-1} \neq \pi_i] \cdot \tau_i^{in} + \mathbb{I}[\pi_i \neq \pi_{i+1}] \cdot \tau_i^{out} \quad (6)$$

$$E_{total}(m) = \sum_{i=1}^n E_i(m), \text{ s.t. } E_i(m) = e_i^{comp} + \mathbb{I}[\pi_{i-1} \neq \pi_i] \cdot e_i^{in} + \mathbb{I}[\pi_i \neq \pi_{i+1}] \cdot e_i^{out} \quad (7)$$

where the τ_i^{comp} and e_i^{comp} are the respective computational latency and energy consumption experienced by L_i given its corresponding mapping, π_i . τ_i^{in} and τ_i^{out} are the latency overhead sustained when loading and writing back graph features from and to the *shared system memory* on the SoC, respectively. The indicator function $\mathbb{I}[\cdot]$ evaluates to 1 only when the associated condition is met; that is, no transmission overhead penalties are sustained between two consecutive layers when they are both assigned the same computing unit. For the energy formula, the same logic of notation applies for every layer L_i .

3.1.3 Mapping Problem Formulation. Define $P(m) = f(T_{total}(m), E_{total}(m))$ to be a combined evaluation function for a mapping configuration m . Let $\mathbb{M}$ be the set of feasible mapping configurations. Then, we can formulate the mapping objective function for an architecture α deployed on a heterogeneous SoC platform as follows:

$$m^* = \max_{m \in \mathbb{M}} P(m), \quad s.t. \quad T_{total} < T_{TRG}, \quad E_{total} < E_{TRG} \quad (8)$$

where the goal is to identify an optimal mapping strategy, m^* , for α such that performance objective function P is maximized with respect to latency and energy under user-specified constraints on latency and energy consumption, T^{TRG} and E^{TRG} , respectively.

3.2 Nested Search Formulation

As the application of graph learning on embedded hardware is a relatively nascent field, the lack of standardization in GNN architectures for edge deployment settings adds another dimension to this design optimization problem. Together with the mapping formulation derived above, a natural question arises as follows: *Given an awareness of the ideal mapping strategy for a GNN onto a heterogeneous MPSoC, can we leverage this information to guide further architectural design optimizations such that the target task accuracy and resource efficiency are enhanced?*

In light of this proposition, we refine our formulation to an *architecture-mapping* co-optimization problem, where the goal is to identify the optimal set of design choices for the GNN architecture and its mapping strategy. Since a Cartesian product of their combined search parameters can result in an enormous search space, we designate two separate subspaces to be managed through a bi-level optimization approach as follows: (a) GNN architecture subspace ($\mathbb{A}$); which describes the set of architectural design choices associated with the GNN model, and (b) Mapping subspace ($\mathbb{M}$); specifying the possible distributed mapping options given the underlying CUs. Through this designation, mapping choices become conditioned on architectural choices, which promotes the generality of this approach. Formally, the nested optimization formulation can be given as follows:

$$\alpha^* = \max_{\alpha \in \mathbb{A}} \psi[Acc(\alpha), P(m^*|\alpha, \mathbb{CU})] \quad (9)$$

$$s.t. \quad m^* = \max_{m \in \mathbb{M}} P(m|\alpha, \mathbb{CU}) \quad (10)$$

where the outer optimization equation targets identifying the optimal set of GNN architectural parameters, α^* , that yield the best scores on a combined function, ψ , of both the accuracy, $Acc(\cdot)$, and performance efficiency $P(\cdot)$. Evaluation of $P(\cdot)$ is contingent upon the results from the inner optimization equation. That is, energy and latency performance evaluations used for scoring a candidate architecture, α , are those obtained for an optimal mapping strategy, m^* . Due to the conflicting nature of the involved objectives, the problem can be solved as a multi-objective optimization providing a Pareto-optimal set of solutions. For instance for the outer optimization objective, an architecture α^* is said to be Pareto-optimal iff for every objective $u \in U$:

$$u_k(\alpha^*) \geq u_k(\alpha) \forall k, \alpha \text{ and } \exists j : u_j(\alpha^*) > u_j(\alpha) \forall (\alpha) \neq (\alpha^*) \quad (11)$$

4 MAGNAS FRAMEWORK

To solve the above GNN *architecture-mapping* co-optimization problem, we present **MaGNAS**, a mapping-aware Graph Neural Architecture Search framework for heterogeneous SoC deployment. **MaGNAS** employs two phases: (i) the construction and training of a ViG supernet to attain a design space of diverse GNN architectural design choices; (ii) the development of a two-tier evolutionary search framework to *identify* optimal *architecture-mapping* pairings.

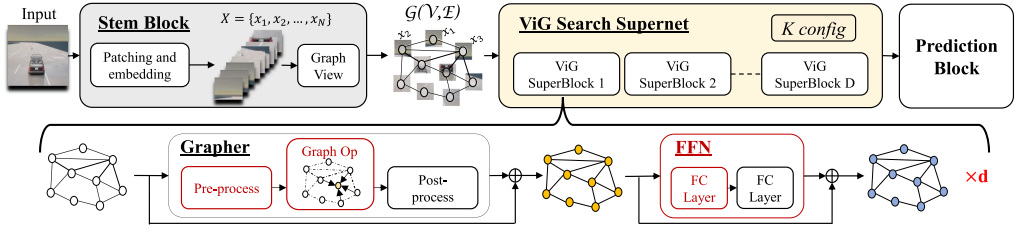


Fig. 2. The ViG supernet implementation for MaGNAS co-search framework. The supernet comprises D ViG search super blocks, each of which constitutes a sequence of d_i Grapher and FFN computing modules. Architectural search parameters characterizing $\mathbb{A}$ subspace are highlighted in red and detailed in the text.

4.1 Supernet Construction and Training

We extend the ViG architecture introduced in Section 2 to construct a supernet of various design choices to characterize an architectural search space $\mathbb{A}$. Briefly, a *supernet* represents a network of networks that can be trained simultaneously to facilitate providing diverse model designs for different deployment scenarios [6]. In the context of ViGs, each subnet within a supernet is defined by a unique set of architectural parameter choices (e.g., choice of GNN layers, #layers, etc.). Additionally, supernets entertain the property of *weight-sharing*, meaning that during the supernet’s training, weight updates for a candidate layer are applied and reused across all subnets that share that particular layer, which enables the simultaneous training of all subnets within it. Once the supernet is trained, a search algorithm can be employed to identify an ideal subnet that meets the target specifications. The ViG supernet is illustrated in Figure 2, where the choice of architectural search parameters for $\mathbb{A}$ is based on observations from both related works [13, 17, 39, 40] as well as from our initial experiments. The supernet construction is detailed in the following:

4.1.1 ViG Superblocks. The backbone ViG-S architecture in [17] comprises 16 computing blocks, each comprising a stack of a *Grapher* and an *FFN* module. On the one hand, characterizing $\mathbb{A}$ on a per-layer or a per-block basis can lead to an explosion in the search space, given the number and cardinality of various search parameters. Conversely, associating the parameters of $\mathbb{A}$ with the entire backbone restricts fine-grained architectural optimizations, not fully exploiting the power of diversified architectural settings at different model stages. As a compromise, we propose *ViG superblocks* to characterize $\mathbb{A}$, where each i^{th} superblock constitutes a collection of d_i ViG blocks sharing the same design choices. Superblocks are inspired by the concept of neural computing blocks in popular architectures (e.g., ResNets), where the same architectural parameter value can be repeated for a stack of consecutive layers. Figure 2 illustrates the composition of our ViG superblock and what architectural parameters are searchable within it. The merits of the ViG superblocks are twofold: (i) they balance the trade-off between architectural diversity and search space complexity; (ii) They facilitate effective management of the depth parameter through d_i while preserving key architectural features.

4.1.2 $\mathbb{A}$ Search Parameters. For each superblock i , we specify the following parameters to construct our architectural search space $\mathbb{A}$:

- *The depth, d_i* , to indicate how many ViG blocks exist in the i^{th} superblock i .
- *Grapher pre-processing* as a binary decision variable to indicate whether a pre-processing layer exists before every graph processing layer.
- *Graph Op* to specify the graph operation employed throughout the i^{th} superblock.

- *FFN module* as a binary decision variable to indicate whether FFN modules should exist in this superblock.
- *FC hidden layer dimension* to specify the size of the intermediate features in the FFN module.

We do not include the Grapher's post-processing layer as part of $\mathbb{A}$ since, in the ViG backbone, it additionally contributes to maintaining the consistency of feature embedding dimensions.

4.1.3 Supernet Training. We train the supernet for our target task using a combination of Cross-Entropy and knowledge distillation loss functions, where for the latter, we employ a pretrained model as a teacher for more representative training on soft labels' training [4, 42]. This training is performed from scratch due to: (i) The ViG is a relatively new GNN architectural concept, and the availability of pretrained weights is still limited, and (ii) loading the exact pretrained model weights from the original ViG backbone [17] can introduce a bias towards certain design choices during training. For instance, the original ViG architecture employed MRConv *Graph Op* throughout the entirety of its graph processing layers. As such, loading their pretrained weights gives MRConv operations an edge over the remaining *Graph Op* choices.

To train the supernet, we sample and train a set of subnets at each iteration. The choice of subnets is realized through 3 separate samplers following the Sandwich sampling rule [42] as follows:

- *Maximum Sampler*: sample the largest subnet from $\mathbb{A}$, that is, the one with the maximum depth and width (i.e., hidden dimension features).
- *Minimum Sampler*: sample the smallest subnet from $\mathbb{A}$.
- *Balanced Sampler*: sample a number of random subnets of different architectural features.

This scheme enables improving the performance of all subnets within the search space simultaneously by pushing the upper and lower performance bounds with every iteration. Furthermore, given how numerous GNN architectures leverage a homogeneous structure, that is, one where the choice of the *Graph Op* is kept consistent throughout the entire architecture, we modify the Maximum/Minimum samplers so that they sample architectures of maximal/minimal sizes, but constituting a randomly selected *Graph Op* repeated throughout the model. This ensures training fairness by pushing the upper and lower boundaries of architectures of different graph operations and avoids inducing a bias towards specific implementations.

4.2 Nested Evolutionary Search: Outer Optimization Engine (OOE)

In order to solve the bi-level *architecture-mapping* optimization problem formulated in Equations (9) and (10), we construct the two-tier evolutionary search framework illustrated in Figure 3 to identify optimal architecture-mapping pairings. Briefly, an evolutionary search is a metaheuristic based on the concept of natural selection in biological evolution, where only the best individuals survive. Specifically, an evolutionary search works by creating a population of candidate solutions from a search space, evaluating each one, and propagating the top-performing solutions to the gene pool of subsequent generations. These solutions can then endure and undergo the genetic operations of mutation and crossover to contribute new derivative solutions for the following generations. This search paradigm is widely used in NP-hard problems to quickly retain optimal solutions while ensuring a broad exploration of gene diversity. In other words, an evolutionary search relies on updating a non-dominated solutions archive with every generation. Thus with each evolution, only new non-dominated solutions from the current population are added, and the newly-dominated ones in the archive are removed.

We first describe the Outer Optimization Engine (OOE), which employs a higher-level evolutionary algorithm whose purpose is to: (i) search through the supernet to identify the most-promising GNN subnets and (ii) rank candidate subnets according to their $Acc(\cdot)$ and $P(\cdot)$ evaluations.

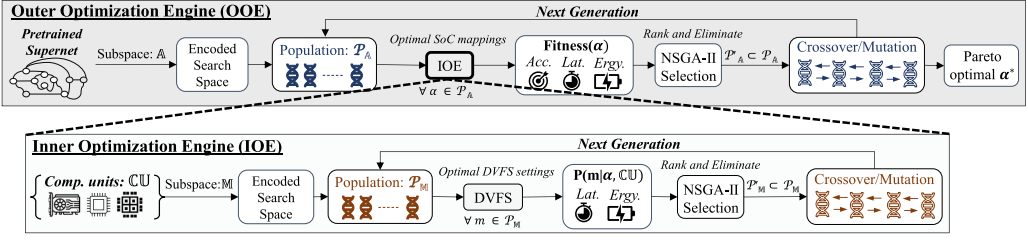


Fig. 3. MaGNAS two-tier evolutionary search framework.

4.2.1 Subspace $\mathbb{A}$ Description. By adopting a Once-For-All (OFA) NAS approach [6], the *training* and *search* stages within MaGNAS are decoupled, significantly reducing the search process overheads as once the supernet has been trained, its search subspace, $\mathbb{A}$, can be reused for the search to identify beneficial subnets. Accordingly, subspace $\mathbb{A}$ in the search stage is encoded as a sequence of 04 discrete vectors, each representing the architectural parameters for each ViG superblock listed in 4.1.2, facilitating the sampling of subnets as GNN architectural design candidates, $\alpha \in \mathbb{A}$.

4.2.2 OOE Evolutionary Search. The next step is to employ a search algorithm to solve the optimization objective in (9) by searching for optimal GNN architectural implementations, α^* . Here, we implemented the NSGA-II evolutionary search algorithm to navigate through $\mathbb{A}$ and explore the subspace of viable design choices. Typically, the search algorithm is run for a pre-specified number of *generations*, where a new population of candidate architectural designs, $\mathcal{P}_{\mathbb{A}}^g$ is sampled with every generation, g . Then, $\forall \alpha \in \mathcal{P}_{\mathbb{A}}^g$, a *fitness* evaluation function, $F(\cdot)$, is applied as follows:

$$F(\alpha) = f(\text{Acc}_{\alpha}, T_{\alpha}, E_{\alpha}) \quad (12)$$

which scores every α based on its target task accuracy, latency, and energy consumption on the target platform denoted by Acc_{α} , T_{α} , and E_{α} , respectively. Acc_{α} evaluation can be obtained directly by evaluating the α model predictive performance on the test dataset, whereas estimates of T_{α} , and E_{α} are provided by the inner optimization engine based on evaluations of the ideal mapping strategy, m^* (which will be detailed in the following subsection). Though we used for $F(\cdot)$ a weighted product function of the objective evaluations in our implementation, we kept its definition here abstract for generality. According to the fitness evaluation scores, every $\alpha \in \mathcal{P}_{\mathbb{A}}^g$ is ranked via the NSGA-II non-dominated sorting algorithm. Based on the rankings, an elimination process is initiated afterward to yield a population subset $\mathcal{P}'_{\mathbb{A}}^g \subset \mathcal{P}_{\mathbb{A}}^g$. Subset $\mathcal{P}'_{\mathbb{A}}^g$ then undergoes *mutation* and *crossover* operations to provide a new population $\mathcal{P}_{\mathbb{A}}^{g+1}$ for the following generation $g + 1$. A uniform mutation is employed on the superblock level by sampling new depth, width, graph operators, etc., under a probability threshold of 0.4. The crossover is applied by randomly picking two individuals from the Pareto set and swapping their superblocks under a probability threshold of 0.5. This iterative search continues until the search budget expires (e.g., a given total number of generations). At the last iteration, a Pareto-optimal set, $\{\alpha^* | m^*\}$, is provided. To provide some perspective based on our experiments, we sample 100 architectures for $\mathcal{P}_{\mathbb{A}}^g$ out of a total $|\mathbb{A}| \approx 2^{29}$ candidates. After fitness evaluations, we select a subset of 30% from the top-ranked candidates as $\mathcal{P}'_{\mathbb{A}}^g$ for the following mutation and crossover processes.

4.3 Nested Evolutionary Search: Inner Optimization Engine (IOE)

To estimate T_{α} and $E_{\alpha} \forall \alpha \in \mathcal{P}_{\mathbb{A}}^g$, we develop an Inner Optimization Engine (IOE) to specify an ideal mapping strategy of α onto the underlying SoC ($\alpha \rightarrow \mathbb{CU}$) and evaluate performance accordingly.

4.3.1 Subspace $\mathbb{M}$ Description. The mapping configuration, m , defined in Equation (5) reflects the encoded discrete vector within the IOE search space that characterizes potential mapping options for each *Grapher* and *FFN* modules from α . We also extend the specification of m in the IOE to incorporate two further mapping options for the *stem* and *prediction* modules (see Figure 2).

4.3.2 IOE Evolutionary Search. Given how the mapping decision space is at least $|\mathbb{CU}|^n$ (see Equation (3)), a brute-force search to determine the ideal mapping, m^* , can be costly. As such, we implement another NSGA-II evolutionary algorithm in the inner optimization level to effectively explore mapping choices within $\mathbb{M}$ and identify the best candidates. Particularly, a population of mapping configurations, denoted by $\mathcal{P}_{\mathbb{M}}^g$, is sampled every generation g by the search algorithm. Then for every $m \in \mathbb{M}$, a fitness evaluation function $P(\cdot)$ is applied as given in the below formula:

$$P(m|\alpha, \mathbb{CU}) = \left(\frac{E_{\alpha}^m}{\max\{E_{\alpha}^{CU}\}} \right)^{\gamma_1} \times \left(\frac{L_{\alpha}^m}{\max\{L_{\alpha}^{CU}\}} \right)^{\gamma_2} \quad \forall \mathbb{CU} \in \mathbb{CU} \quad (13)$$

where E_{α}^m and L_{α}^m are the respective energy and latency sustained by α when its components are deployed onto the underlying hardware following a mapping strategy m . Each of these values is then normalized by the best *standalone* deployment option from $\mathbb{CU}$, denoted here by E_{α}^{CU} and L_{α}^{CU} , respectively. The reasons for this normalization are twofold: (i) To ensure fairness when comparing various mapping options for α ; (ii) To enforce achieving comparable, if not improved, performance scores over those obtained by the canonical standalone deployment options. For instance, if mapping the entirety of α onto a *GPU* component is the best option with respect to latency, then all latency evaluations are normalized by L_{α}^{GPU} . γ_1 and γ_2 are user-specified tunable hyperparameter values to enable prioritizing one performance objective or the other. For our experiments, we constructed accessible lookup tables by benchmarking computing blocks of varying architectural configurations onto the target CUs, allowing low-overhead estimations of latency and energy during the search.

Based on these evaluations, another non-dominated sorting algorithm is instantiated to rank mapping configurations, retaining the top-ranked configurations to provide population subset $\mathcal{P}_{\mathbb{M}}^{g'} \subset \mathcal{P}_{\mathbb{M}}^g$. Afterwards, subset $\mathcal{P}_{\mathbb{M}}^{g'}$ undergoes mutation and crossover to provide $\mathcal{P}_{\mathbb{M}}^{g+1}$ as the new population for the next generation. The mutation is uniformly applied by flipping the CU for each GNN computing block under a probability threshold of 0.4. The crossover is applied by randomly selecting two individuals from the Pareto set and interchanging their CUs mapping under a probability threshold of 0.8. Once the search budget expires, $E_{\alpha}^{m^*}$ and $L_{\alpha}^{m^*}$ are returned as evaluations for the best configuration, m^* , to be used for E_{α} and T_{α} in the OOE, respectively.

4.3.3 Constrained Search. To support specifying L_{TRG} and E_{TRG} as search constraints during the search procedure as in Equation (8), we designate an additional option for the selection procedure of the IOE non-dominated sorting algorithm to filter out mapping options from $\mathcal{P}_{\alpha}^m$ that do not conform to the pre-specified constraints, allowing only compliant mapping options to proceed to the next stage of mutation and crossover. If there were no compliant mappings, the standalone evaluations are returned for E_{α} and T_{α} . In general, L_{TRG} and E_{TRG} can also be instated at the selection process of the OOE, where α architectures whose E_{α} and T_{α} do not meet target performance scores are eliminated from the population before the OOE's mutation and crossover stage.

4.3.4 Performance Characterization. Generally, estimates of E_{α}^m and L_{α}^m for every $m \in \mathcal{P}_{\mathbb{M}}^g$ can be provided through a multitude of approaches (e.g., predictive models). As was shown in Equation (4), the dimensional consistency of graph features offered throughout the ViG backbone has led to a tractable space of evaluation possibilities, enabling the construction of low-cost lookup tables to directly retrieve performance estimates of various architecture-mapping configurations.

Simply put, the lookup tables are indexed by the architectural parameters of a computing block, L_i , and the CU to whom it is mapped. By invoking the tables for every block in α given m , the performance overheads of each block can be aggregated to estimate the total E_α^m and L_α^m . Although lookup tables work for our case, proxy prediction models can be more feasible for a different GNN architecture in which the graph features dimensions change as a result of inconsistent graph structures.

4.3.5 DVFS Search Support. We also include the option to supplement $\mathbb{M}$ subspace with the configuration setting choices of dynamic voltage and frequency scaling (DVFS) features. Predominantly, numerous standard heterogeneous SoC components integrate this feature to support a diverse set of operational modes serving different execution contexts, as in to enable switching between *low-power* and *high performance* modes. Here, to better capture the fine-grained effects of altering DVFS settings, we specify a DVFS search block in the IOE as a *third* optional optimization level contingent upon the choices of m and α . This is convenient as the search space of the DVFS is small compared to $\mathbb{A}$ and $\mathbb{M}$ and does not incur as much search overhead. In typical real-time operational contexts, DVFS settings are kept the same across all the computing blocks of α . This made a direct brute-force search through DVFS options sufficient to identify configurations that maximize the IOE fitness score in objective (13). Formally, if we denote a single set of DVFS configuration settings as ϑ and the overall DVFS search space as Ψ , then the DVFS search objective is given as:

$$\vartheta^* = \max_{\vartheta \in \Psi} P(m|\alpha, \mathbb{C}\mathbb{U}, \vartheta) \quad (14)$$

where the performance evaluation of m becomes also contingent upon the choice of $\vartheta \in \Psi$.

5 EXPERIMENTS

In this Section, we conduct extensive experiments, in-depth analysis, and ablation studies using a real MPSoC platform and hardware simulation on four(04) state-of-the-art image classification datasets to assess the merit of MaGNAS in designing ViG architectures and mapping them onto heterogeneous CUs, as well as its ability to scale with an increasing degree of problem complexity.

5.1 Experimental Setup

5.1.1 Supernet Design. We build our supernet on top of the ViG-S variant [17] with 16 computing blocks, each a *Grapher* and an *FFN* block. We group every four (04) computing blocks into a *ViG superblock*, and assign to each K nearest neighbor values of 12, 16, 20, and 24, respectively, which enables aggregation of features from farther nodes with each superblock. To support dynamic width and depth configurations, we transform each ViG superblock into a *slimmable* neural network following [41]. To support varying graph operations, we specify a dynamic graph processing layer in the *Grapher* with four concurrent branches reflecting different GCN operational choices for *Graph Op*: (1) EdgeConv [31], (2) GIN [34], (3) GraphSAGE [16], and (4) Max-Relative GraphConv [23]. As mentioned in Section 4.1, the GNN search space also includes options to skip the *Grapher*'s pre-processing layer and the entirety of the *FFN* module throughout a given ViG superblock.

5.1.2 Datasets and Training. We employ four (04) image classification datasets of CIFAR-10, CIFAR-100, Tiny-Imagenet, and Oxford-Flowers. To transform the images to graphs, images are first scaled to $224 \times 224 \times 3$ resolution, and transformed through the *Stem* block into a graph of nodes $N = 196$, each of dimension $D = 14 \times 14 \times 320$. The supernet training for each dataset is run for 150, 150, 250, and 250 for each respective dataset in the order in which they were stated. The training is performed using an Adam optimizer with a momentum of 0.9, weight decay of

0.05, and dropout set to 0.2. We use cosine as a learning rate scheduler with an initial LR of 0.003 and batch size of 320 on a cluster of 20 GPUs of Nvidia RTX 2080 Ti (11 GB).

5.1.3 Evolutionary Search Settings.

Table 1 lists the search sub-spaces of $\mathbb{A}$, $\mathbb{M}$, and Ψ designated within our optimization framework. For the optimization process, we fix the population size to 100 and 200 and the number of generations to 50, and 10 for the OOE and IOE, respectively. We adopt uniform mutation and crossover with respective probabilities of 0.8 and 0.4. We employ a dynamic encoding scheme in which the IOE evolutionary algorithm changes the size of the genome vector -for the mapping strategy encoding- according to the architectural parameters of the sampled GNN to avoid sampling meaningless decision variables (e.g., mapping choices for skipped FFN and FC-pre layers). Combining the OOE and IOE, we explored $\sim 1.6 \times 10^6$ candidates of GNN architectures and deployment settings on an Nvidia Xavier AGX platform. The search process takes around ~ 1 – 2 GPU days to complete, depending on the complexity of the accuracy evaluation for each dataset.

5.1.4 Hardware Experimental Settings. We evaluate our approach using two hardware experimental setups presenting a variety of computing units and architectural features: (i) NVIDIA Jetson AGX Xavier [1], as a real target MPSoC platform; (ii) MAESTRO [20, 21], as a hardware simulator tool.

① **NVIDIA Jetson AGX Xavier:** We employ the NVIDIA Jetson AGX Xavier MPSoC [1] as our primary experimental testbed. The platform is equipped with a high-performance Volta GPU of 512 GPU cores and 64 Tensor cores, and an energy-efficient DLA. We specify both components as the usable computing units of $\mathbb{C}\mathbb{U}$ and characterize them as the feasible deployment options of $\mathbb{M}$. Both components share the same 16 GB 256 bits LPDDR4x 136.5 GB/s system memory and are orchestrated by the same CPU NVIDIA Carmel Arm 64 bits. To run workloads on GPU/DLA, we use the TensorRT 8.4 compiler running on top of CUDA 11.4 and cuDNN 8.3.2. As TensorRT is limited by the set of operations that can be executed on DLA, we consider this limitation in our performance characterization by enabling the *GPU fallback* feature for the non-supported operations. The AGX Xavier also supports hardware reconfiguration of the clock frequencies of CPU, GPU, EMC, and DLA to emulate different hardware settings and power budgets, which we use to implement the DVFS search space Ψ . Unless otherwise stated, performance evaluations in our experiments are performed under the high-performance DVFS setting (MaxN).

② **MAESTRO:** For the hardware scalability analysis, we leverage the MAESTRO tool [20, 21] to simulate a use-case of an SoC with three (03) heterogeneous CUs, where the heterogeneity is expressed by varying the dataflow configuration on each accelerator given how different neural network workloads exhibit different affinities towards dataflow choices for maximizing performance efficiency. For example, a weight stationary dataflow (like *kcp_ws* from MAESTRO and that of the DLA accelerator in the Nvidia Xavier) maximizes filter weights' reuse which is useful for layers whose same filters are used to compute multiple outputs, limiting the number of times weights need to be fetched from the main memory and improving energy efficiency in the interim [9]. We use the native dataflows in MAESTRO of *kcp_ws*, *ykp_os*, and *dpt* for our 3 CUs, which for simplicity, we denote by **DSA-k**, **DSA-y**, and **DSA-d**. We also use for this experiment the PyramidViG-M architecture detailed in the following.

Table 1. Search Space Parameters for GNN Architectures

Decision variables	Values	Cardinality
Supernet Search Space ($\mathbb{A}$)		
Superblock depth (d)	{2, 3, 4}	3
Graph Op	{Max-Relative, EdgeConv, GraphSAGE, GIN}	4
Skip pre-process (fc_use)	{False, True}	2
Skip post-process (ffn_use)	{False, True}	2
FFN hidden features (w)	{96, 192, 320}	3
Mapping Search Space ($\mathbb{M}$) for NVIDIA Xavier AGX		
Computing units	{GPU, DLA}	2
Mapping granularity	{Stem, Grapher, FFN, Cls}	$O(1.7 \times 10^{12})$
DVFS Settings Search space (Ψ) for NVIDIA Xavier AGX		
CPU clock frequency	{1728MHz, 2265MHz}	2
GPU clock frequency	{520MHz, 900MHz, 1377MHz}	3
EMC clock frequency	{1065MHz, 2133MHz}	2
DLA clock frequency	{1050MHz, 1395MHz}	2

5.1.5 Baselines. The efficacy of our approach is assessed regarding the following GNN architectural and hardware mapping baseline:

- ① **GNN architectures baselines:** These include the original isotropic ViG-S model in [17] as well as its variants by altering *Graph Op* (i.e., the GCN operation) where the *Graph Op* remains consistent across all the layers. Specifically, we identify the baselines by their recurring *Graph Op* operation: (1) **b0**: ViG-S/Max-Relative, (2) **b1**: ViG-S/EdgeConv, (3) **b2**: ViG-S/GIN, and (4) **b3**: ViG-S/GraphSage. For the scalability analysis of the IOE, we also consider the PyramidViG-M as the alternative ViG backbone that sustains graph features dimensional reductions as the network deepens. We implemented PyramidViG-M to follow the feature dimensional reductions across stages as in [17] and fixed four (04) blocks within each superblock in the supernet (recall 5.1.1).
- ② **HW-mapping baselines:** We consider the default *standalone* deployment options – i.e., the full mapping of an entire ViG model to a singular CU (e.g., to the GPU only). We also consider hybrid mapping strategies in which inter-CU transitions are limited, as proposed in [10].
- ③ **MAESTRO GNN baseline:** We use the aforementioned PyramidViG-M GIN-variant for our hardware scalability experiments using the MAESTRO simulator. For the convenience of MAESTRO, we define the GIN operation by its low-level implementations of the *aggregation* and *combination* phases. That is, the *aggregation* entails a matrix multiplication between the adjacency matrix and the feature embedding matrix, whereas the *combination* entails another matrix multiplication to transform the aggregated graph features to another representation for the following layer.

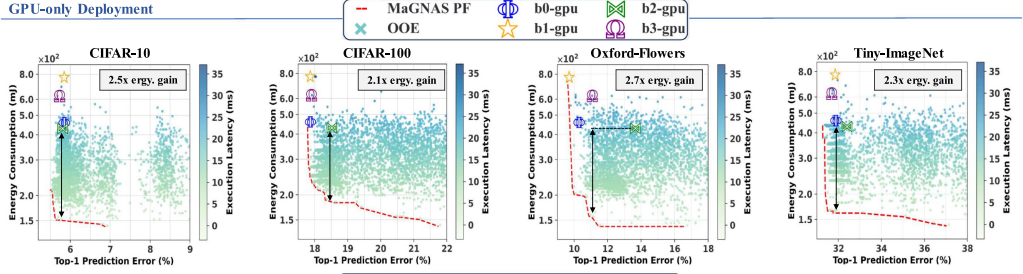
5.2 OOE Results: GNN Architecture Optimization

We first examine the merit of the OOE in identifying GNN architectures that can achieve favorable performance trade-offs compared to the baselines. In Figure 4, the first two rows depict the explored GNN architectures from $\mathbb{A}$ by the OOE on the four (04) datasets given *standalone* mapping strategies on GPU-only (*top row*) and DLA-only (*middle row*). Compared to the baselines defined above, our obtained Pareto-optimal GNN architectures generally dominate all baselines on the four image classification datasets with regard to the three performance metrics of accuracy, latency, and energy consumption. Specifically, the OOE can identify GNN architectures that achieve up to $\sim 3.6\times$ latency speedup than baselines when deployed onto the GPU; can realize up to $\sim 2.8\times$ more energy efficiency gains compared to the baselines when deployed onto the DLA – all while maintaining comparable accuracy scores. As will be emphasized in the subsequent Section 5.4, the reasons for this dominance by the OOE’s GNN architectures is attributed to the allowed diversification of *Graph Op* across the different ViG superblocks (as specified in $\mathbb{A}$ from Table 1), which has facilitated achieving better accuracy-performance trade-offs. Moreover, skipping the *FFN* and the *Grapher*’s FC pre-processing layers offers attractive design choices to avoid unnecessary computation, especially when the set of features is limited and can be already captured by the basic layers of the *Grapher* modules – which is the case for the simpler datasets (e.g., CIFAR-10). Our OOE recognized this property and leveraged its knowledge to concentrate its search on identifying GNN architectural parameters that achieve the best accuracy levels with the minimal number of *FFN* and FC pre-processing layers.

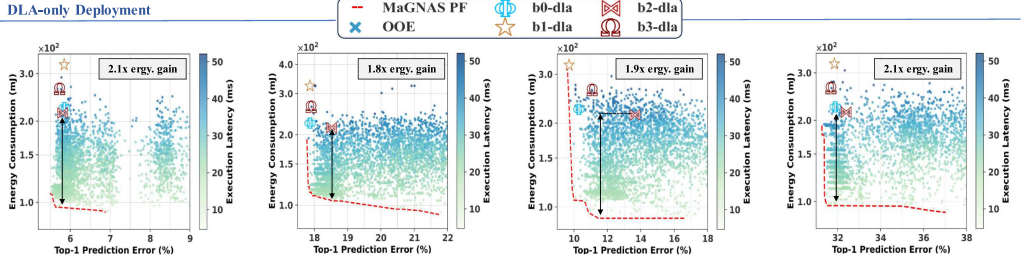
5.3 IOE Results: Hardware Mapping Optimization

We further assess the efficacy of the IOE in identifying effective mapping configurations for provided GNN architectures. The bottom row of Figure 4 shows the optimization results when exploring mapping strategies from $\mathbb{M}$ for the *top-performing* GNN architectures (as ranked by Equation (12)) provided to the IOE. The results are reported for CIFAR-10 and grouped by TOP-1

GPU-only Deployment



DLA-only Deployment



GPU-DLA Distributed Deployment

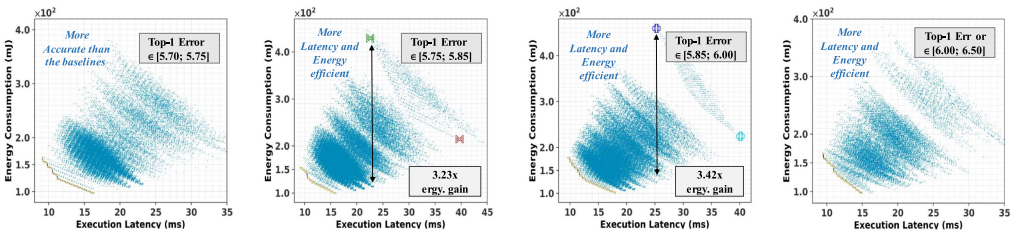


Fig. 4. The first two rows show the performance of the explored GNNs in (A) by the OOE on four datasets (from left to right: (a) CIFAR-10, (b) CIFAR-100, (c) Oxford-Flowers, and (d) Tiny-ImageNet). The Hardware metrics (i.e., latency and energy) are shown for *GPU-only* deployment in the first row and for *DLA-only* deployment in the second row. The third row shows the IOE results on CIFAR-10 grouped by prediction error intervals.

error intervals in each sub-figure. A similar trend has also been observed in the other datasets. At each top-1 error interval, we can observe that the IOE explored various mapping strategies, as illustrated by the latency-energy trade-offs. The bulk of these trade-offs are captured within the range of performance values from the standalone deployment options, that is, between the GPU-only and DLA-only mapping options' latency/energy consumption values, as depicted by the middle sub-figures. Remarkably, the explored configurations form distinguishable contours, each showing a specific GNN architecture alongside its explored mapping options – represented by the different latency-energy trade-off values. Specifically, the GPU-only and DLA-only mapping configurations for each GNN architecture are located at the boundaries of its curved line. The intermediate points between the extremes illustrate the performance of the distributed deployment settings and show how each mapping configuration results in different latency-energy trade-offs.

Furthermore, as both *GNNs* and *mappings* are considered together in the IOE design space, superior energy gains can be realized through more compact GNN architectures. For instance, as illustrated in the third sub-Figure, an energy gain up to $\sim 3.42\times$ can be attained compared to the **b2-gpu** while preserving comparable latency and accuracy levels by opting for another GNN architecture and distributed mapping. Upon comparing the curve lines, we can observe that GNN

Table 2. Detailed Performance Results, GNN Architectural Parameters, and Mapping Strategies of our Pareto Optimal Models (**a0-a3**)

Datasets	GNN Models	TOP-1 Acc (%)	Graph-Ops (M, E, G, S)	FFN-use (%)	FC pre-use (%)	Latency (ms)	Energy (mj)	GPU-use (%)	DLA-use (%)
All-datasets	Baseline-b0	C10: 94.15, C100: 82.13 F: 89.71, Ti: 68.12	M-M-M-M	100	100	G: 25.28 D: 40.11	G: 459.44 D: 224.41	-	-
	Baseline-b1	C10: 94.15 C100: 82.13 F: 90.29, Ti: 68.15	E-E-E-E	100	100	G: 33.74 D: 62.11	G: 770.36 D: 323.70	-	-
	Baseline-b2	C10: 94.20, C100: 81.49 F: 86.37, Ti: 67.62	G-G-G-G	100	100	G: 22.49 D: 39.62	G: 429.07 D: 214.35	-	-
	Baseline-b3	C10: 94.27, C100: 82.10 F: 88.92, Ti: 68.32	S-S-S-S	100	100	G: 29.57 D: 57.77	G: 623.76 D: 263.48	-	-
CIFAR-10 (C10)	Ours-a0	94.25	G-G-G-G	25	25	16.02	97.0	09	91
	Ours-a1	94.46	G-G-G-G	100	0	19.49	118.00	17	83
	Ours-a2	94.32	G-M-G-G	25	0	11.19	121.14	75	25
	Ours-a3	94.32	G-M-G-G	25	0	14.18	105.11	33	67
CIFAR-100 (C100)	Ours-a0	82.13	S-G-S-G	100	25	17.72	180.56	50	50
	Ours-a1	82.17	S-S-S-S	100	75	34.72	271.62	30	70
	Ours-a2	81.63	G-G-G-G	50	50	15.06	131.81	50	50
	Ours-a3	82.13	S-G-S-G	100	25	17.29	197.80	55	45
Oxford-Flowers (F)	Ours-a0	89.90	M-G-M-M	75	75	14.37	153.54	69	31
	Ours-a1	88.43	G-G-G-G	0	50	9.60	119.07	90	10
	Ours-a2	88.43	G-G-G-G	0	50	12.30	105.88	40	60
	Ours-a3	89.02	M-G-G-G	25	25	12.82	116.63	50	50
Tiny-ImageNet (Ti)	Ours-a0	68.40	M-G-G-G	25	0	13.07	114.89	50	50
	Ours-a1	68.40	M-G-G-G	25	0	15.47	102.06	17	83
	Ours-a2	68.51	M-G-G-G	75	25	16.37	122.56	38	62
	Ours-a3	68.51	M-G-G-G	75	25	17.87	115.78	19	81

The original ViG-S and its variants (**b0-b3**) on the four datasets on the NVIDIA Jetson Xavier AGX SoC platform. 'G' and 'D' in the latency and energy columns indicate GPU and DLA, respectively.

architectures that outperformed the baselines in the OOE (i.e., in the standalone deployment options shown by the extremes) typically maintain their dominance within the IOE and proves that rank is preserved across GNN architectures and mapping schemes in this joint search space.

5.4 Analysis of Pareto Search and Models

5.4.1 Results Discussion. In Table 2, we provide a detailed analysis of performances, architectural parameters, and mapping strategies of the ViG baselines [**b0-b3**] and a selection of our final Pareto optimal models from the two-tier search [**a0-a3**] for each dataset. As shown, although our models maintain comparable accuracy scores to the baselines, they generally achieve better speedups and energy efficiency results. To be more precise, our models achieve on average $\sim 1.57\times$ and $\sim 2.49\times$ latency speedups; $\sim 3.38\times$ and $\sim 1.65\times$ more energy efficiency when compared against the original ViG baseline fully-deployed onto the GPU and DLA, respectively. This dominance is primarily attributed to 3 factors: (i) the enabled diversification of Graph Op parameter throughout the ViG superblocks, which enables interleaving both *powerful* and *resource-efficient* operators within a model architecture. For instance, examining the Oxford-Flowers results in the Table, model **a0** interleaves both Max-Relative and GIN operators. The former contributes to the model's representational capacity and compensates for the inadequacy of GIN operators in capturing long-range dependencies from the graph nodes features, ultimately leading the model to surpass baseline **b0**'s accuracy score (89.9% to 89.71%). On the other hand, the employment of GIN operator – alongside other factors – leads **a0** to achieve superior latency and energy efficiency scores. (ii) The additional varying architectural parameters from $\mathbb{A}$ (e.g., FFN-use) enable tuning the model's size and learning capacity to the task and dataset complexity. (iii) The distributed mapping strategies, as indicated by the *GPU-use* and *DLA-use* columns in Table 2, further balance the latency-energy trade-offs by effectively utilizing different CUs.

5.4.2 Hypervolume and Pareto Composition Analysis. To appraise the efficiency of our nested evolutionary search algorithm in identifying meaningful and mapping configurations, we compare its Hypervolume [27] against those of baseline OOE searches conducted on the *standalone*

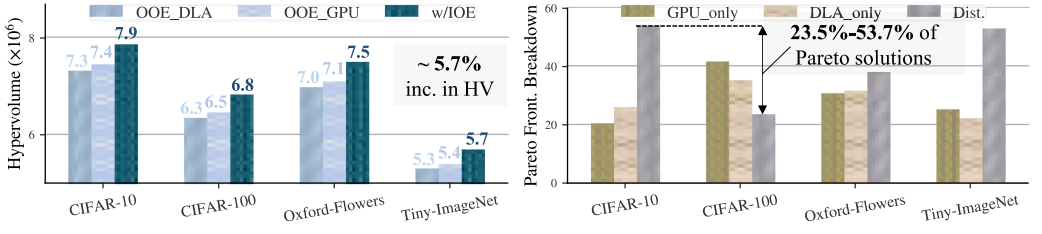


Fig. 5. *Left*: Hypervolume analysis when including the IOE against those of the standalone OOE for the DLA and GPU. *Right*: Breakdown of the combined Pareto Fronts constituents on the basis of mapping options.

Table 3. Details and Comparison of the GNN Workload Assignment

Mapping option	Stem	Grapher	FFN	Cls	#transit	Lat.	Enrg.
DLA-only	D	D-D-D-D-D-D-D-D	D-D-D-D-D-D-D-D	D	0	25.56	121.74
GPU-only	G	G-G-G-G-G-G-G-G	G-G-G-G-G-G-G-G	G	0	13.42	273.22
constr-transit1	D	D-G-G-G-G-G-G-G	D-G-G-G-G-G-G-G	G	1	16.31	232.60
constr-transit1	G	G-G-G-G-G-D-D-D	G-G-G-G-G-D-D-D	D	1	17.42	226.79
constr-transit2	D	D-G-G-G-G-G-G-D	D-G-G-G-G-G-G-D	D	2	17.58	220.23
constr-transit2	G	G-G-D-D-D-G-G-G	G-G-D-D-D-G-G-G	G	2	17.11	227.15
Ours (IOE)	D	G-G-G-G-G-G-G-G	G-D-D-D-D-G-D-D	D	12	17.29	197.8

‘G’ and ‘D’ indicate GPU and DLA assignment, respectively. Note that each Grapher block is first succeeded by a corresponding FFN block.

deployment options on the GPU and DLA. Succinctly, the Hypervolume measures the volume of the dominated area in the objective space by the estimated Pareto fronts. In Figure 5 (left), we can observe that the nested search (w/IOE) improves the Hypervolume scores over the baseline OOE_GPU search by ~5.7% on average across the four (04) datasets, indicating the IOE’s merit in extending the dominated area in the search space. In Figure 5 (right), we complement the Hypervolume analysis with a breakdown of the Pareto front composition with regard to the mapping strategies. Specifically, we consider the non-dominated solutions by combining Pareto fronts obtained at every generation. As seen, the *distributed* mapping options constitute 23.5%–53.7% of the solutions on the Pareto front, indicating their value in elevating resource efficiency for the various models.

5.4.3 Analysis of GNN Workload Distribution. In this subsection, we showcase how different GNN workload assignments across the GPU and DLA influence the latency-energy tradeoffs. In Table 3, we select one of the Pareto-optimal models, **Ours-a3** on CIFAR-100, and compare three mapping configurations: (i) Standalone options in which the model is fully deployed on either GPU or DLA. (ii) Constrained transition options (as introduced in [10]) where the number of allowable inter-CU transitions is limited to those that offer the best tradeoffs in order to mitigate data transmission overheads (i.e., the write-back and initial cold cache misses). (iii) Ours (IOE) are the mapping options provided through our IOE with unconstrained inter-CU transitions.

As no single optimal solution exists for any distributed mapping strategy, we ensure a fair comparison between our approach and the constrained transition strategies by comparing evaluations of one objective function (energy) while fixing the other (latency). As such, for each constrained transition option, we use two (02) Pareto optimal solutions whose latency values are closest to our solution – i.e., solutions with latency closest to 17.29 ms. From the reported results in Table 3, we can observe that with our unconstrained mapping strategy, a single inference sustains 197.8 mJ on average, which is more efficient than the best energy numbers, 226.79 mJ and 220.23 mJ, experienced by each of the other distributed mapping baselines, ‘constr-transit1’ and ‘constr-transit2’, respectively. The reasons for this improvement can be attributed to the following: (i) graph feature sizes are relatively small throughout the ViG models, leading to low inter-CU transmission

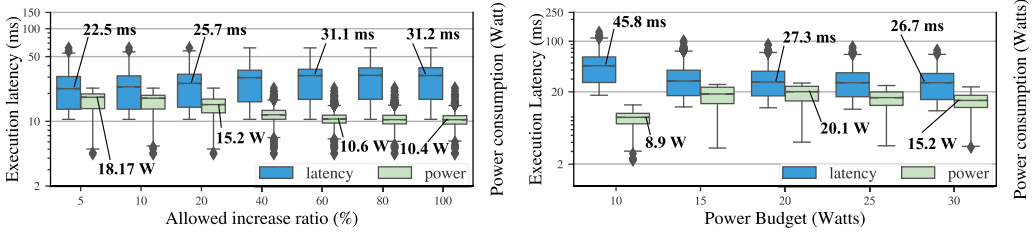


Fig. 6. Results of the two constrained optimization: Latency and power consumption numbers are reported under variation of (Left) the allowable latency increase ratio compared to GPU-only, and (right) the available power budget. Numbers indicate median values.

overhead penalties to be experienced on the Xavier SoC. As Such, our IOE optimization strategy was able to exploit this property to identify more efficient mapping configurations with a larger number of transitions. (ii) Each computing block type within the ViG exhibits different affinities towards the underlying CUs. Thus, our IOE optimization strategy leveraged the other property of unconstrained transitions to map as many Grapher blocks to the GPU as feasible and as many FFN blocks to the DLA as possible before transmission costs become non-negligible.

5.5 Constraint-aware Optimization

As many embedded systems employ real-time execution requirements, we test the effectiveness of our framework when the search algorithm is performed under strict latency and power constraints. In particular, we specify two experiments, each associated with one of the following constraints: **(1) Latency**, in which the constraint specifies the max allowable increase in latency compared to the *standalone* deployment option on the fastest SoC component (i.e., GPU-only). **(2) Power budget**, by fixing low values of clock frequencies and a limited number of CPU cores and memory speed transmission [2]. The first constraint is common for real-time systems governed by strict execution deadlines, whereas the second constraint is more common for battery-powered systems operating on limited power budgets. We conduct the two constrained optimization on the IOE using baselines [b0-b3] and our models [a0-a3] on the CIFAR-100. We report the absolute latency and average power consumption values in Figure 6. We also characterize the latency constraint by enforcing a max *allowable increase ratio* from the fastest CU (i.e., the GPU). As shown in left Figure 6, low latency increase ratio ($\leq 20\%$) leads the IOE towards delegating more computation kernels to the GPU, resulting in more power-demanding mapping strategies. Compared to the soft-constraint case (i.e., w/ tolerance of 100% increase in latency), the power demands at an allowed increase ratio of 5% are $1.75\times$ more.

As the tolerable increase ratio rises ($\geq 30\%$), the constraint on the search is gradually relaxed. As shown in Table 4, the optimizer gains more freedom in exploring mapping options and favors delegating more computation kernels to the DLA for energy efficiency. The power efficiency gains start to plateau around a 50% increase ratio, indicating that the IOE has converged onto mapping strategies that maximize the fitness formula (as defined in (13)) by balancing latency and power efficiency. This convergence is sensible given how between the GPU and DLA, one component is roughly twice as effective as the other with regards to one performance objective, i.e., execution latency on the GPU is almost $2\times$ less than the DLA, and the DLA incurs $2\times$ less power consumption than the GPU (see Table 2); given that we assigned equivalent weights for the objectives in

Table 4. Workload Distribution

Workload Distribution	Allowable latency increase ratio (%)						
	5	10	20	40	60	80	100
Avg. GPU utilization	0.97	0.91	0.74	0.56	0.50	0.50	0.50
Avg. DLA utilization	0.03	0.09	0.26	0.44	0.50	0.50	0.50

the fitness evaluation formula in (13), i.e., $\gamma_1 = \gamma_2 = 1$. The second experiment depicted in the right Figure 6 shows that at tighter power budget constraints, the IOE focuses on identifying power-efficient mapping options at the expense of a slight decrease in latency, resulting in mappings that assign more GNN workloads to the DLA as depicted in Table 5. We note that in this experiment, we also maintain the latency minimization as objective, which also explains the low DLA utilization ratio values reported in Table 5. For instance, to satisfy the 10 Watts power constraint, the IOE specifies mapping settings with a median latency of $45.8\% - 1.71\times$ more than the latency experienced at a power budget of 30 Watts. More latency-efficient mappings are identified with refined workload distribution as the power budgets are relaxed.

Table 5. Workload Distribution

Workload Distribution	Available Power Budget (mW)				
	10	15	20	25	30
Avg. GPU utilization	0.74	0.76	0.88	0.88	0.81
Avg. DLA utilization	0.26	0.24	0.13	0.13	0.19

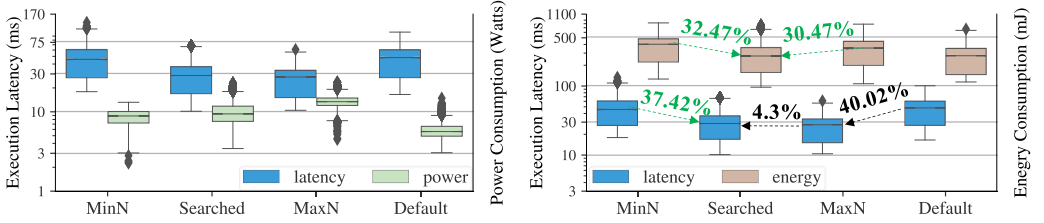


Fig. 7. Ablation on the impact of including DVFS optimization within the IOE. **Searched** DVFS is compared against the **MinN**, **MaxN**, and **Default** settings with regards to (Left): Latency-Power trade-offs, and (Right): Latency-Energy trade-offs. Numbers in the right Figure indicate percentage change in values.

5.6 Ablation Study on the Impact of DVFS

In this experiment, we assess the merit of including DVFS optimization within the IOE. We reuse the baselines [b0-b3] and our models [a0-a3] from the CIFAR-100 experiment. Their mappings are kept fixed, and we run the models through the DVFS optimization engine to assess how performance can be further enhanced. Specifically, we consider the following DVFS settings: (i) **MaxN**, which resembles the high-performance mode on the Jetson Xavier SoC with clock frequencies set to the maximum. (ii) **MinN**, which is an opposing best-effort mode for low-power operation in which clock frequencies are set to the minimum. (iii) **Searched**; in which DVFS settings are searchable within the IOE (see Table 1 for the values). iv) **Default**; in which we use the default dynamic DVFS heuristic with CPU and GPU governors fixed to *Schedutil*, *nvhos*t *podgov*, respectively. In this last setting, clock frequencies are dynamically adjusted at runtime depending on the underlying resources utilization, where clock frequencies are ranged from 0 to the maximum value on each component. We note that in addition to the GPU and DLA frequency variations, we also scale the CPU and EMC clock frequencies as both influence data transmissions between the shared system memory and private memories of GPU/DLA. We run the IOE with the same optimization parameters to ensure a fair evaluation. In Figure 7, we illustrate the performance trade-offs as incurred by the explored (*GNN architectures* $\times$ *HW mappings*) under the 04 DVFS settings. As expected, the left subfigure shows that the Searched mode exhibits a balanced trade-off between latency and power compared to the MinN and MaxN modes. More interestingly, however, the Searched setting is able to identify configurations that yield superior energy gains to the fixed DVFS modes. In particular, the median latency and energy consumption values of Searched are **37.42%** and **32.47%** less than MinN, respectively. On the other hand, though Searched incurs a **4.3%** increase in its median latency compared to MaxN, it can achieve an order of magnitude more energy savings reaching **30.47%**. This implies that the IOE identified the DVFS as a viable tuning knob to enhance energy efficiency by scaling clock frequencies across the different components. Moreover, latency

in Searched is improved by **40.02%** compared to the default DVFS governor. This is explained by the underlying logic of the dynamic heuristic, which only considers the hardware utilization and overlooks workload properties such as computation and memory requirements. For instance, memory-bounded workloads may benefit from GPU/DLA core downscaling with reduced energy at the same latency level. These properties are captured in our Search mode as we adjust the frequencies according to the GNN and mapping configurations. In addition, The default governors are set to avoid the idle state when the computing unit is not used, by lowering the frequency to 0, which helps in minimizing the power consumption (as shown in the left subfigure) but also worsens the execution latency as computing units usually need a warm-up stage to operate steadily after swapping between low and high frequencies.

5.7 Generality and Scalability

Employing an evolutionary algorithm (EA) for the IOE may seem excessive when the backbone ViG architecture is an isotropic one that does not experience feature map sizes change and when the mapping is performed across merely 02 CUs. As such, we perform an additional set of experiments in which we assess the scalability and generality of the IOE on the search-space levels of: (i) **the ViG architectural backbone**; where the supernet's backbone is implemented as a pyramid variant that allows dimensional reductions from one superblock to the next (recall Figure 2), unlike the aforementioned isotropic counterpart, and (ii) **the hardware CUs**; by simulating a case with 03 heterogeneous CUs. The details are provided below.

5.7.1 On the ViG Architectural Level. Using the Nvidia Xavier SoC with GPU and DLA, we compare the mapping results from the IOE between the isotropic (ViG-S) and pyramid (PyramidViG-M) variants (recall Section 5.1.5). As we analyze the effectiveness of the inner EA, we fix the GNN from the OOE for both variants by setting the design parameters, $\mathbb{A}$, in Table 1 (i.e., $d=4$, $Graph\ Op=GIN$, $fc_use=False$, $ffn_use=False$, $w=192$), and specify an optimization budget of 2×10^4 evaluations.

As depicted in Figure 8, we can observe in the left subfigure that for the isotropic ViG, the explored mapping options follow well-defined spaced patterns between the two mapping extremes of *GPU-only* and *DLA-only*, offering almost uniform linear trade-offs between the energy efficiency and execution latency across various mapping options on the Pareto front. This results from the Grapher and FFN blocks being replicated throughout an isotropic architecture. As such, the performance evaluation of the different mapping options becomes predominantly influenced by the percentage of Grapher/FFN blocks assigned to each CU, irrespective of their order. Given such a setting, a scalarization method can be sufficient to determine the Pareto front by varying the ratio of mappable workloads on either CU. However, for the PyramidViG on the right, this property does not hold as each Grapher/FFN block entertains different dimensions of their input and output features depending on its position, leading to varying performance characterizations. As such, we observe that the sampled mapping options are more diverse in their energy and latency characterizations and that the Pareto front exhibits stronger convexity than its isotropic counterpart, reflecting a diverse, more complex mapping space.

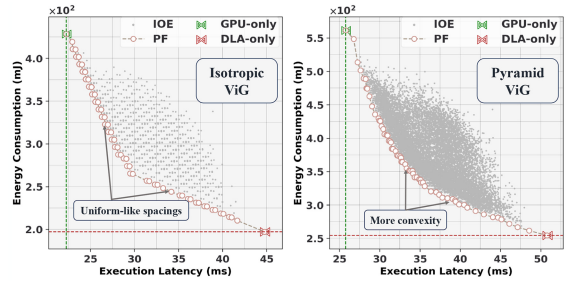


Fig. 8. The results of the IOE EA optimization on the Isotropic Vision GNN (left) and Pyramid Vision GNN (right).

5.7.2 On the Hardware CU Level. Using the PyramidViG-M, we investigate how MaGNAS scales when the search space is further compounded with an increasing number of viable CUs. We simulate such use-case using MAESTRO tool [21] to specify 3 DSAs of diverse dataflows for CU heterogeneity (see the details in 5.1.4). As every layer within MAESTRO is defined via low-level implementations (including *aggregation* and *combination* layers), we can characterize processing overheads within PyramidViG-M on a *layerwise* basis and combine them to characterize larger blocks (e.g., Grapher). At this point, we find that each ‘layer’ rather than ‘block’ can exhibit different performance characteristics at different ViG stages. For instance, the *aggregation* sustains a substantial overhead when processing the sizable graph feature matrices at earlier blocks. This is predominantly due to the DSAs in MAESTRO not being implemented initially to support graph acceleration – similar to how numerous SoC platforms (e.g., the Xavier) do not widely integrate specialized graph acceleration engines. As such, we can simulate an additional case to study the mapping on a *layerwise* granularity to assess further how the EA in the IOE scales when the number of mappable options dramatically increase. To provide context, the mapping space of the PyramidViG-M is $O(1.72 \times 10^{12})$ in the *blockwise* using 2 CUs; $O(1.67 \times 10^{16})$ in the *blockwise* using 3 CUs; and $O(1.67 \times 10^{23})$ in the *layerwise* 3 CUs case, indicating an increasing level of problem complexity. In Figure 9, we demonstrate how the inner

EA scales effectively as the search space is expanded from the *blockwise* to the *layerwise* mapping granularity. We first specify a fixed optimization budget of 6×10^4 evaluations for both. Moreover, although fully deploying the architecture on DSA-d completely dominates DSA-k deployment, the latter is still included since it represents the optimal mapping option for some individual layers. In the *blockwise* case (left), we observe that the EA focuses on exploring more mapping solutions at the energy consumption extremes due to coarse-grained characterization of the Grapher block, leading it to identify distributed mapping options that dominate the standalone extreme, i.e., the EA identifies a distributed mapping configuration that achieves 1.25× energy gains over DSA-y for the same latency level. The opposite occurs for the *layerwise* search, where despite the much larger optimization space, the EA was capable of recognizing benefits from distributing the *aggregation* and *combination* across different DSAs, leading it to concentrate the search more at the centralized latency-energy trade-off region. For example, at execution latency of $\sim 2.2 \times 10^8$ cycles, the *layerwise* search by the IOE was able to identify a mapping option that incurs 28.6 mJ compared to 31.9 mJ from the *blockwise* search.

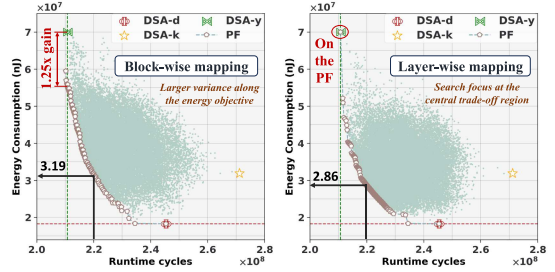


Fig. 9. The results of the IOE optimization on MAESTRO [21] with: (i) *Block-wise mapping granularity* (left) and (ii) *Layer-wise mapping granularity* (right).

5.7.3 On the Power of Evolution. We further analyze the hypervolume improvement when using an EA compared to a random search. We fix an optimization budget of 5000 evaluations for each and showcase the results in Figure 10 at different evolution stages for the mapping onto 3 CUs experiment. Normalized by a maximum achievable value from our previous results, we observe that the normalized hypervolume in the

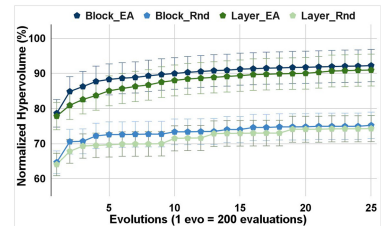


Fig. 10. Evolutionary Vs. Random search.

Figure reaches $\sim 92\%$ improvement for the EA compared to $\sim 75\%$ for the random search. We also notice that both *blockwise* and *layerwise* converge to proximate values despite the larger gap at the earlier evolutions (i.e., generations), further indicating the EA's capacity to scale.

6 DISCUSSION AND FUTURE DIRECTIONS

① **Key Takeaways.** Hardware-software design optimizations and workloads mappings are increasingly studied in the literature [4, 5, 10, 11]. What distinguishes this work is its specialization in considering the details of: (i) GNNs' computational flow irregularity; (ii) workload distribution across heterogeneous CUs with varying degrees of support for graph operators. Furthermore, ViG is a relatively emergent class of GNNs, and there remains room for improvement along the design, characterization, and training of ViG supernets, which can only improve as the application of ViGs – and GNNs in general – at the Edge continues to proliferate. All things considered, MaGNAS has demonstrated encouraging results that can help pave the way for future lines of research.

② **Generality and scalability.** In analyzing the generality of MaGNAS (Section 5.7), we have demonstrated the heterogeneity of hardware accelerators through diversifying dataflows across HW accelerators. In practice, heterogeneity can also occur through varying other factors such as processing engines per accelerator, shared buffer size, off-chip memory bandwidth, etc., all of which can influence the hardware efficiency of the workloads. MaGNAS has been shown capable of generalizing to the different forms of heterogeneity as it relies on high-level performance characterization that abstracts underlying hardware compositions. Furthermore, experiments on real SoCs with different HW accelerators and levels of heterogeneity from that of the Nvidia Xavier is still needed to corroborate that MaGNAS can scale effectively to diverse platforms.

③ **Graph operation support limitations.** As MAESTRO does not natively support the sparse matrix multiplications, we implemented GNN operations within the simulator as generic matrix multiplications, which has led to considerable execution overheads for the *aggregation* phase regarding latency and energy. This is indeed a situation akin to the case when GNN workloads are to be run on generic, uncustomized edge devices that lack proper support for specialized accelerators for GNN operations. In such cases, mapping optimizations can be particularly beneficial in mitigating the impact of such hardware deficiencies. Furthermore, as GNNs grow in popularity, promising steps are being taken towards developing new dataflows for reconfigurable spatial accelerators to support irregular graph computational sequences, which will also bring about the need for new architectural simulators to effectively model their performance overheads.

④ **Other Application Domains.** Vision-based applications provided practical, tangible use case motivations for the *GNNs-on-SoCs* scenario, and accordingly, they have become the primary target application of this work. With that being said, the manner in which MaGNAS has been developed enables it to generalize to other emerging applications on edge SoCs that employ GNNs for their primary computational workloads. For instance, the support for mapping on both the blockwise and layerwise levels of granularity within MaGNAS enables it, with some fine-tuning, to serve other types of emerging GNN-based applications at the edge by maximizing GNNs' efficiency across a broad range of diverse CUs integrated onto the same chip.

7 RELATED WORKS

① **GNNs for vision.** Through learning graph-level features, GNNs achieved remarkable performance on a variety of computer vision tasks, such as activity recognition [37] and point clouds classification [22, 31]. Traditionally, the success of GCNs in computer vision applications relied on the graph construction technique, which in many cases was tailored to suit the input data semantics and downstream task. Scene graph generation [25, 33, 43] emerged as a viable approach to generate a graph of objects and their relations from an image through cascading an object detector

and a GCN model. The ViG [17], a generic architecture upon which our framework is constructed, represents a standard GCN backbone to generate and process graphs from raw images to serve general computer vision applications.

② **Hardware acceleration for GNNs** The two phases of GNN favor different classes of accelerators: GNN acceleration favors MIMD architectures to address the irregularity of graph operations by providing high random access memory bandwidth and small data access sizes, whereas DNN acceleration is achieved through SIMD architectures for exploiting data locality through caches or local scratchpads. As such, numerous works [3, 7, 19, 29, 36, 39] have proposed hybrid accelerator architectures comprising separate engines and specialized hardware components to effectively manage the non-uniform GNN dataflow on both an *intra*- and *inter-phase* level. However, such proposed accelerator designs are acutely specialized ASICs, complicating their integration into numerous commodity hardware platforms and SoCs. Since GNNs are becoming increasingly popular, recent research efforts [15] have directed their approach towards characterizing the design space of dataflow choices to enable running GNNs on customary reconfigurable spatial accelerators, intending to identify convenient dataflows to service various GNN use cases. The philosophy behind our method follows the latter trend. However, it is complementary to both approaches since it abstracts the underlying accelerator architecture and adds another layer of design space exploration to characterize joint search space of GNN architectures and the inter-phase pipelining across heterogeneous computing components in an SoC.

③ **Distributed Computing of GNNs.** Distributing DNN workloads across the heterogeneous computing resources of CPU, GPU, DLAs, and FPGAs, is an active field of research [5, 10, 18, 26, 35]. Researchers have recently explored how to distribute GNN workloads to enhance performance efficiency by exploiting the underlying heterogeneous hardware composition via task-level, data-level, and pipelining forms of parallelism [8]. For instance, the work in [44] proposed to decouple GNNs onto CPU-FPGA heterogeneous platform to speedup GNN inference.

④ **Graph Neural Architecture Search.** Recent research works investigated how to leverage the power of Neural Architecture Search to automate the design process of GNNs. Earlier works adopted search approaches like Reinforcement Learning [13, 14, 47] or Evolutionary algorithms [28]. The work in [40] further proposed a generalized GNNs' design space with a knowledge distillation method from GNN model-task pairs. However, these approaches mostly fall under the training-in-the-loop NAS category. Furthermore, limited or no awareness of the underlying hardware computing platform capabilities was taken. As such, more recent works in [45, 46] proposed to move towards the once-for-all approach [6], which employs a supernet that characterizes the design space of the GNN architectures. Specifically, the training of the supernet can be conducted only once by leveraging the property of weight-sharing. On the hardware side, [45] adopts a co-design NAS approach for GNN and hardware accelerator, whereas [46] optimizes the GNN design to suit underlying commodity edge computing platforms. Our work falls under the same category

Table 6. Comparison between Related Graph Neural Architecture Search Works and Ours

	[14]	[13]	[47]	[28]	[45]	[46]	MaGNAS
Training-in-the-loop NAS	✓	✓	✓	✓			
Once-for-all NAS					✓	✓	✓
Vision GNN							✓
Hardware Awareness					✓	✓	✓
GNN-Hardware co-design					✓		
Edge Computing Setting						✓	✓
Distributed Mapping							✓

of HW-aware NAS for GNNs as these two. However, several features distinguish this work from others: (i) our supernet is designed to consider the emerging class of vision-based GNNs (ViGs); (ii) support for evaluating candidate ViG subnets during the search process based on their best mapping options that leverage pipelining parallelism across diverse computing units within the MPSoC edge platform; (iii) our two-tier search algorithm implementation allows the inner optimization engine to be extensible to other MPSoCs and GNN supernets serving other tasks. We summarize the differences in Table 6.

8 CONCLUSION

In this paper, we presented MaGNAS, a mapping-aware Graph Neural Architecture Search framework for the distributed deployment of vision GNN onto heterogeneous SoCs. MaGNAS characterizes a GNN architectural design space bound with prospective mapping options, enabling the identification of model designs optimized to the distributed deployment scheme. MaGNAS employs a two-tier evolutionary search framework to identify optimal *architecture* and *mapping* pairings that provide the best performance trade-offs. Extensive experimentation, in-depth analysis, and ablation studies using a real MPSoC platform and hardware simulation have showcased the merit of MaGNAS in designing ViG architectures and mapping them onto heterogeneous MPSoCs.

ACKNOWLEDGEMENT

This work was supported by the National Science Foundation (NSF) under award CCF-2140154.

REFERENCES

- [1] [n.d.]. NVIDIA Jetson AGX Xavier Delivers 32 TeraOps for New Era of AI in Robotics. <https://developer.nvidia.com/blog/nvidia-jetson-agx-xavier-32-teraops-ai-robotics/>
- [2] [n.d.]. Power management and clock frequency scaling. <https://docs.nvidia.com/jetson/archives/r34.1/DeveloperGuide/text/SD/PlatformPowerAndPerformance.html>
- [3] Adam Auten, Matthew Tomei, and Rakesh Kumar. 2020. Hardware acceleration of graph neural networks. In *2020 57th ACM/IEEE Design Automation Conference (DAC'20)*. IEEE, 1–6.
- [4] Halima Bouzidi, Mohanad Odema, Hamza Ouarnoughi, Smail Niar, and Mohammad Abdullah Al Faruque. 2023. HADAS: Hardware-aware dynamic neural architecture search for edge performance scaling. In *Design, Automation & Test in Europe Conference & Exhibition (DATE'23)*.
- [5] Halima Bouzidi, Mohanad Odema, Hamza Ouarnoughi, Smail Niar, and Mohammad Abdullah Al Faruque. 2023. Map-and-conquer: Energy-efficient mapping of dynamic neural nets onto heterogeneous MPSoCs. In *Proceedings of the 60th ACM/IEEE Design Automation Conference (DAC)*.
- [6] Han Cai et al. 2019. Once-for-all: Train one network and specialize it for efficient deployment. In *International Conference on Learning Representations (ICLR'19)*.
- [7] Cen Chen, Kenli Li, Xiaofeng Zou, and Yangfan Li. 2021. Dygnn: Algorithm and architecture support of dynamic pruning for graph neural networks. In *2021 58th ACM/IEEE Design Automation Conference (DAC'21)*. IEEE, 1201–1206.
- [8] Chaoqi Chen and others. 2022. A survey on graph neural networks and graph transformers in computer vision: A task-oriented perspective. *arXiv preprint arXiv:2209.13232* (2022).
- [9] Yu-Hsin Chen, Joel Emer, and Vivienne Sze. 2016. Eyeriss: A spatial architecture for energy-efficient dataflow for convolutional neural networks. *ACM SIGARCH Computer Architecture News* 44, 3 (2016), 367–379.
- [10] Ismet Dagli et al. 2022. AxoNN: Energy-aware execution of neural network inference on multi-accelerator heterogeneous SoCs. In *Proceedings of the 59th ACM/IEEE Design Automation Conference (DAC'22)*.
- [11] Nael Fasfous et al. 2022. Anaconga: Analytical hw-cnn co-design using nested genetic algorithms. In *2022 Design, Automation & Test in Europe Conference & Exhibition (DATE'22)*. IEEE, 238–243.
- [12] Brian Gaide et al. 2019. Xilinx adaptive compute acceleration platform: VersalTM architecture. In *Proceedings of the 2019 ACM/SIGDA International Symposium on Field-Programmable Gate Arrays*. 84–93.
- [13] Yang Gao et al. 2020. Graph neural architecture search.. In *IJCAI*, Vol. 20. 1403–1409.
- [14] Yang Gao, Hong Yang, Peng Zhang, Chuan Zhou, and Yue Hu. 2020. Graph neural architecture search. In *Proceedings of the Twenty-Ninth International Joint Conference on Artificial Intelligence, IJCAI-20*. 1403–1409.
- [15] Raveesh Garg et al. 2022. Understanding the design-space of sparse/dense multiphase GNN dataflows on spatial accelerators. In *2022 IEEE International Parallel and Distributed Processing Symposium (IPDPS'22)*. IEEE, 571–582.

- [16] Will Hamilton, Zhitao Ying, and Jure Leskovec. 2017. Inductive representation learning on large graphs. *Advances in Neural Information Processing Systems* 30 (2017).
- [17] Kai Han, Yunhe Wang, Jianyuan Guo, Yehui Tang, and Enhua Wu. 2022. Vision GNN: An image is worth graph of nodes. In *Advances in Neural Information Processing Systems*.
- [18] Jangryul Kim and Soonhoi Ha. 2022. Energy-aware scenario-based mapping of deep learning applications onto heterogeneous processors under real-time constraints. *IEEE Trans. Comput.* (2022).
- [19] Kevin Kinningham et al. 2022. GRIP: A graph neural network accelerator architecture. *IEEE Trans. Comput.* (2022).
- [20] Hyoukjun Kwon et al. 2019. Understanding reuse, performance, and hardware cost of dnn dataflow: A data-centric approach. In *Proceedings of the 52nd Annual IEEE/ACM International Symposium on Microarchitecture*. 754–768.
- [21] Hyoukjun Kwon, Prasanth Chatarasi, Vivek Sarkar, Tushar Krishna, Michael Pellauer, and Angshuman Parashar. 2020. Maestro: A data-centric approach to understand reuse, performance, and hardware cost of dnn mappings. *IEEE Micro* 40, 3 (2020), 20–29.
- [22] Loic Landrieu and Martin Simonovsky. 2018. Large-scale point cloud semantic segmentation with superpoint graphs. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*. 4558–4567.
- [23] Guohao Li, Matthias Muller, Ali Thabet, and Bernard Ghanem. 2019. Deepgcns: Can gcns go as deep as cnns?. In *Proceedings of the IEEE/CVF International Conference on Computer Vision*. 9267–9276.
- [24] Shih-Chieh Lin, Yunqi Zhang, Chang-Hong Hsu, Matt Skach, Md E. Haque, Lingjia Tang, and Jason Mars. 2018. The architectural implications of autonomous driving: Constraints and acceleration. In *Proceedings of the Twenty-Third International Conference on Architectural Support for Programming Languages and Operating Systems*. 751–766.
- [25] Arnav Vaibhav Malawade, Shih-Yuan Yu, Brandon Hsu, Deepan Muthirayan, Pramod P. Khargonekar, and Mohammad Abdullah Al Faruque. 2022. Spatiotemporal scene-graph embedding for autonomous vehicle collision prediction. *IEEE Internet of Things Journal* 9, 12 (2022), 9379–9388.
- [26] Roger Pujol, Hamid Tabani, Leonidas Kosmidis, Enrico Mezzetti, Jaume Abella Ferrer, and Francisco J Cazorla. 2019. Generating and exploiting deep learning variants to increase heterogeneous resource utilization in the nvidia xavier. In *31st Euromicro Conference on Real-Time Systems (ECRTS'19)*, Vol. 23.
- [27] Ke Shang, Hisao Ishibuchi, Linjun He, and Lie Meng Pang. 2020. A survey on the hypervolume indicator in evolutionary multiobjective optimization. *IEEE Transactions on Evolutionary Computation* 25, 1 (2020), 1–20.
- [28] Min Shi, Yufei Tang, Xingquan Zhu, Yu Huang, David Wilson, Yuan Zhuang, and Jianxun Liu. 2022. Genetic-gnn: Evolutionary architecture search for graph neural networks. *Knowledge-Based Systems* 247 (2022), 108752.
- [29] Jacob R. Stevens et al. 2021. GNNerator: A hardware/software framework for accelerating graph neural networks. In *2021 58th ACM/IEEE Design Automation Conference (DAC'21)*. IEEE, 955–960.
- [30] Emil Talpes et al. 2020. Compute solution for tesla's full self-driving computer. *IEEE Micro* 40, 2 (2020), 25–35.
- [31] Yue Wang, Yongbin Sun, Ziwei Liu, Sanjay E. Sarma, Michael M. Bronstein, and Justin M. Solomon. 2019. Dynamic graph cnn for learning on point clouds. *Acm Transactions On Graphics (tog)* 38, 5 (2019), 1–12.
- [32] Zonghan Wu, Shirui Pan, Fengwen Chen, Guodong Long, Chengqi Zhang, and S. Yu Philip. 2020. A comprehensive survey on graph neural networks. *IEEE Transactions on Neural Networks and Learning Systems* 32, 1 (2020), 4–24.
- [33] Danfei Xu, Yuke Zhu, Christopher B. Choy, and Li Fei-Fei. 2017. Scene graph generation by iterative message passing. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*. 5410–5419.
- [34] Keyulu Xu, Weihua Hu, Jure Leskovec, and Stefanie Jegelka. 2019. How powerful are graph neural networks?. In *International Conference on Learning Representations*.
- [35] Lei Xun et al. 2020. Optimising resource management for embedded machine learning. In *2020 Design, Automation & Test in Europe Conference & Exhibition (DATE)*. IEEE, 1556–1561.
- [36] Mingyu Yan, Lei Deng, Xing Hu, Ling Liang, Yujing Feng, Xiaochun Ye, Zhimin Zhang, Dongrui Fan, and Yuan Xie. 2020. Hygen: A gen accelerator with hybrid architecture. In *2020 IEEE International Symposium on High Performance Computer Architecture (HPCA'20)*. IEEE, 15–29.
- [37] Sijie Yan, Yuanjun Xiong, and Dahua Lin. 2018. Spatial temporal graph convolutional networks for skeleton-based action recognition. In *Proceedings of the AAAI Conference on Artificial Intelligence*, Vol. 32.
- [38] Jianwei Yang, Jiasen Lu, Stefan Lee, Dhruv Batra, and Devi Parikh. 2018. Graph r-cnn for scene graph generation. In *Proceedings of the European Conference on Computer Vision (ECCV'18)*. 670–685.
- [39] Haoran You et al. 2022. Gcod: Graph convolutional network acceleration via dedicated algorithm and accelerator co-design. In *2022 IEEE International Symposium on High-Performance Computer Architecture (HPCA'22)*. IEEE, 460–474.
- [40] Jiaxuan You, Zhitao Ying, and Jure Leskovec. 2020. Design space for graph neural networks. *Advances in Neural Information Processing Systems* 33 (2020), 17009–17021.
- [41] Jiahui Yu et al. 2019. Slimmable neural networks. In *International Conference on Learning Representations*.
- [42] Jiahui Yu et al. 2020. Bignas: Scaling up neural architecture search with big single-stage models. In *Computer Vision—ECCV 2020: 16th European Conference, Glasgow, UK, August 23–28, 2020, Proceedings, Part VII* 16. Springer.

- [43] Shih-Yuan Yu, Arnav Vaibhav Malawade, Deepan Muthirayan, Pramod P. Khargonekar, and Mohammad Abdullah Al Faruque. 2021. Scene-graph augmented data-driven risk assessment of autonomous vehicle decisions. *IEEE Transactions on Intelligent Transportation Systems* 23, 7 (2021), 7941–7951.
- [44] Bingyi Zhang et al. 2022. Low-latency mini-batch GNN inference on CPU-FPGA heterogeneous platform. In *2022 IEEE 29th International Conference on High Performance Computing, Data, and Analytics (HiPC'22)*. 11–21.
- [45] Yongan Zhang et al. 2021. G-CoS: Gnn-accelerator co-search towards both better accuracy and efficiency. In *2021 IEEE/ACM International Conference On Computer Aided Design (ICCAD'21)*. IEEE, 1–9.
- [46] Ao Zhou et al. 2023. Hardware-aware graph neural network automated design for edge computing platforms. In *Proceedings of the 60th ACM/IEEE Design Automation Conference (DAC'23)*.
- [47] Kaixiong Zhou et al. 2022. Auto-gnn: Neural architecture search of graph neural networks. *Frontiers in Big Data* (2022).

Received 23 March 2023; revised 2 June 2023; accepted 30 June 2023