GCoDE: Efficient Device-Edge Co-Inference for GNNs via Architecture-Mapping Co-Search

Ao Zhou, Jianlei Yang, *Senior Member, IEEE*, Tong Qiao, Yingjie Qi, Zhi Yang, Weisheng Zhao, *Fellow, IEEE*, Chunming Hu

Abstract—Graph Neural Networks (GNNs) have emerged as the state-of-the-art graph learning method. However, achieving efficient GNN inference on edge devices poses significant challenges, limiting their application in real-world edge scenarios. This is due to the high computational cost of GNNs and limited hardware resources on edge devices, which prevent GNN inference from meeting real-time and energy requirements. As an emerging paradigm, device-edge co-inference shows potential for improving inference efficiency and reducing energy consumption on edge devices. Despite its potential, research on GNN device-edge co-inference remains scarce, and our findings show that traditional model partitioning methods are ineffective for GNNs. To address this, we propose GCoDE, the first automatic framework for GNN architecture-mapping Co-design and deployment on Device-Edge hierarchies. By abstracting the device communication process into an explicit operation, GCoDE fuses the architecture and mapping scheme in a unified design space for joint optimization. Additionally, GCoDE's system performance awareness enables effective evaluation of architecture efficiency across diverse heterogeneous systems. By analyzing the energy consumption of various GNN operations, GCoDE introduces an energy prediction method that improves energy assessment accuracy and identifies energy-efficient solutions. Using a constraint-based random search strategy, GCoDE identifies the optimal solution in 1.5 hours, balancing accuracy and efficiency. Moreover, the integrated co-inference engine in GCoDE enables efficient deployment and execution of GNN co-inference. Experimental results show that GCoDE can achieve up to $44.9\times$ speedup and 98.2% energy reduction compared to existing approaches across diverse applications and system configurations.

Index Terms—Graph Neural Networks, Device-Edge Co-Inference, Neural Architecture Search, Edge Devices, System Awareness

1 Introduction

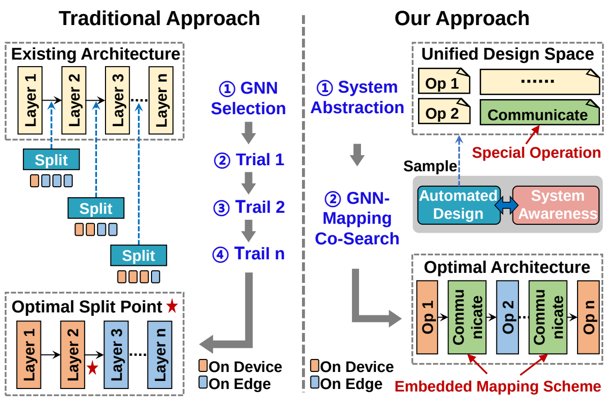
As edge devices become more intelligent and deep learning breakthroughs continue, the demand for deploying models to process various collected data in real-time on the device side is growing [1, 2]. However, limited hardware resources make it challenging to meet latency and energy requirements when deploying complex deep learning models [3]. In particular, Graph Neural Networks (GNNs) have recently excelled in processing irregular data structures, making them a popular choice for graph-related applications in edge scenarios, such as point cloud processing [4] and natural language processing [5]. Additionally, the rising popularity of various sensors in mobile devices also encourages the deployment of GNNs to the wireless network edge for real-time sensing and interaction. For instance, autonomous drones require immediate obstacle detection from point clouds [6], where the latency of cloud communication is unacceptable. Likewise, executing speech-based interaction locally for smart assistants [7] is crucial for safeguarding user privacy. However, the significant computational cost of GNNs and the limited hardware resources on edge devices pose major challenges to meeting these strict real-time and energy requirements, severely limiting their application in real-world edge scenarios. This is demonstrated by deploying the popular point cloud processing model DGCNN [8] on a Raspberry Pi 3B, which achieves less than **0.3** fps, far below practical requirements (typically above 30 fps [9]).

Research efforts have been made to address the inefficiency of GNNs on edge devices. [4, 10] reduced GNN computation by manually simplifying the model structure. Meanwhile, HGNAS [11] and [12] adopted a more efficient hardware-aware neural architecture search (NAS) approach to design hardware-friendly GNNs for edge devices. Despite these performance improvements, GNN acceleration remains limited by constrained hardware resources. For example, the hardware-efficient GNNs developed by HGNAS [11] only increase point cloud processing speed to 2 fps on the Raspberry Pi, as reported in their paper. To address resource constraints, device-edge co-inference has emerged as a promising approach for deploying models at the network edge [13, 14]. In this paradigm, choosing an appropriate split point to partition the model into device-side and edge-side execution parts can significantly reduce inference latency and on-device energy consumption. However, most device-edge co-inference methods are designed for DNNs [15, 16], with few tailored to the unique computational patterns of GNNs.

To explore the device-edge co-inference paradigm for GNNs, Branchy-GNN [17] introduces a multi-branch de-

- *This work is supported in part by the National Natural Science Foundation of China (Grant No. 62072019), the Fundamental Research Funds for the Central Universities, the Beijing Natural Science Foundation (Grant No. L243031), and the National Key R&D Program of China (Grant No. 2023YFB4503704 and 2024YFB4505601). Corresponding authors are Jianlei Yang and Chunming Hu.*
- *A. Zhou, J. Yang, T. Qiao, Y. Qi, and C. Hu are with School of Computer Science and Engineering, Beihang University, Beijing, China. Email: jianlei@buaa.edu.cn, hucm@buaa.edu.cn.*
- *Z. Yang is with School of Computer Science and Engineering, Peking University, Beijing, China.*
- *W. Zhao is with School of Integrated Circuits and Engineering, Beihang University, Beijing, China.*

Manuscript received on Sept 7, 2024, revised on Aug 11, 2025.



🟥: Low 🟩: High	Traditional	Ours
Granularity	Layer Split	Operation Mapping
Scalability	🟥	🟩
Design Efficiency	🟥	🟩
Awareness	🟥	🟩
Performance	🟥	🟩

Fig. 1. Device-Edge Co-Inference for GNNs.

sign methodology that manually partitions the DGCNN model to minimize communication costs. However, this approach neglects hardware characteristics and fails to explore novel architectures, resulting in limited performance gains. In practice, simply partitioning existing models does not achieve the desired efficiency, and leveraging device-edge co-inference for GNNs remains challenging. First, the intermediate data generated during GNN inference includes features and potentially graph-structured data, which must be carefully considered to balance communication and computation overheads. Second, GNNs exhibit a hybrid computation pattern involving computation-intensive matrix operations and memory-intensive graph processing. Each has distinct hardware sensitivities on heterogeneous devices, requiring an effective system-aware approach. Additionally, separating architecture design from mapping often leads to sub-optimal performance, necessitating a co-optimized approach for both. Furthermore, there is currently no effective way to perceive on-device energy consumption, which is critical for edge applications. Therefore, an elegant co-inference methodology tailored for GNNs is needed.

To address the above-mentioned challenges, this paper proposes a novel NAS-based automated design and deployment framework for GNN device-edge co-inference, named GCoDE. Given user requirements, GCoDE can efficiently search and deploy optimal GNN architectures with their concomitant mapping schemes for target systems, achieving both accuracy and efficiency under latency and energy constraints. Specifically, GCoDE achieves joint optimization of GNN architectures and their operation mappings within the device-edge hierarchy in a single exploration. This is based on an ingenious idea of **treating the inter-device communication process during co-inference as a special GNN operation** and incorporating it into the GNN design space. Moreover, leveraging this system abstraction, an efficient GNN-based performance awareness approach is integrated to guide the constraint-based search process. Furthermore, GCoDE develops a specialized device-edge co-inference engine for GNNs to support flexible operation mapping execution. Fig. 1 provides an intuitive comparison between traditional approaches and GCoDE. The *scalability* refers to the ability of the framework to adapt and maintain performance across diverse system configurations, whereas *design efficiency* denotes the reduction in manual effort and design cycle time. Unlike traditional methods that manually split existing models, GCoDE integrates automated design with system awareness to jointly optimize GNN architectures and operation mappings, thereby substantially enhancing scalability, design efficiency, and overall performance.

Our main contributions are summarized as follows:

- We propose an architecture and operation mapping scheme co-search framework dubbed GCoDE which largely resolves the inefficiency of GNN device-edge co-inference. To the best of our knowledge, GCoDE is the first automated design and deployment framework for GNNs in the device-edge co-inference paradigm, opening up an exciting perspective for exploring much more efficient GNN solutions.
- We propose a unified GNN design space for device-edge co-inference, enabling the joint optimization of architecture and operation mapping in a single constraint-based exploration, balancing accuracy, latency, and on-device energy consumption.
- To the best of our knowledge, GCoDE is also the first to achieve system performance awareness for GNNs in heterogeneous co-inference systems. The proposed latency and energy predictors achieve up to 85.3% and 70.1% accuracy within a 10% error bound across different system configurations.
- We provide a comprehensive analysis of on-device energy consumption during GNN co-inference and extend the awareness dimensions to develop an energy prediction method that significantly improves accuracy over existing estimation methods.
- Extensive experiments on diverse applications and system configurations consistently validate the effectiveness of our GCoDE framework, e.g., GCoDE can achieve up to $44.9\times$, $21.6\times$ speedups and 98.2%, 96.2% on-device energy savings over DGCNN and the existing SOTA GNN Device-Edge co-inference approach Branchy-GNN, respectively, while maintaining the same accuracy.

The rest of the paper is organized as follows. Section 2 discusses the related works. Section 3 presents the motivations and provides an overview of the proposed framework. Section 4 details the architecture–mapping co-exploration method, and Section 5 describes the system performance awareness approach. Section 6 introduces the device–edge deployment engine. Section 7 reports the experimental results and analysis. Finally, Section 8 concludes the paper.

2 Related Work

In this section, we discuss the relevant prior work from three aspects. The comparison of support features between GCoDE and other frameworks is illustrated in Tab. 1.

2.1 Graph Neural Networks

GNN inference, based on the message-passing paradigm, typically involves two distinct operations: *Aggregate* and

TABLE 1
Comparison of support features

Supported Features	GCoDE	HGNAS [11]	MaGNAS [18]	Branchy [17]
Design Automation	✓	✓	✓	✗
Architecture Exploration	✓	✓	✓	✗
Performance Awareness	✓	✓	✓	✗
▷ Single Device	✓	✓	✗	✗
▷ Heterogeneous	✓	✗	✓	✗
▷ Heterog. Wireless Edge	✓	✗	✗	✗
Multi-Objective Optimization	✓	✓	✓	✗
Device-Edge Deployment	✓	✗	✗	✓
Runtime Optimization	✓	✗	✗	✗

Combine [19]. The former aggregates features from source neighbors for each vertex, while the latter updates each vertex's feature using a weight matrix. Additionally, for some edge applications, such as point cloud processing, the *sample* operation is included to extract the graph structure from raw data for subsequent computation. To accelerate GNN inference on edge devices, [10] proposes reusing the *sample* results across GNN layers, while [4] introduces simplified message construction during neighbor aggregation. These manual optimization efforts require extensive trial-and-error on the target device, resulting in weak scalability and lengthy design cycles. Moreover, GraNNite [20] focuses on hardware-aware optimization of GNN inference for resource-constrained NPUs in client devices through a multi-stage approach, achieving significant performance gains. Unlike these works, which target single-device acceleration on specific hardware, GCoDE addresses the broader device–edge co-inference scenarios across heterogeneous environments by jointly optimizing GNN architectures and operation mappings with system performance awareness, enabling efficient and scalable GNN deployment at the wireless network edge.

2.2 Device-Edge Co-Inference

Device-edge co-inference is an emerging paradigm for edge AI applications that effectively addresses the constrained on-device resource and limited bandwidth present in device-only and edge-only inference [16]. The common approach to co-inference is to split a standard network into two parts and deploy them on a device and an edge server, respectively. Extensive studies have explored various splitting methods and collaborative mechanisms, greatly improving the inference efficiency of DNN models in edge applications like IoT [15]. Nevertheless, the research for GNNs is still lacking. Branchy-GNN [17] (denoted as Branchy) was the first co-inference method for GNNs, improving inference efficiency by manually splitting the existing model. Unfortunately, the lack of hardware awareness and separate design manner prevented it from realizing the full potential of this paradigm. In contrast, GCoDE integrates system performance awareness with architecture-mapping co-search, ensuring adaptation to various system configurations.

2.3 Graph Neural Architecture Search

GNN NAS has been proposed to design application-customized GNNs, addressing the inefficiencies of manual

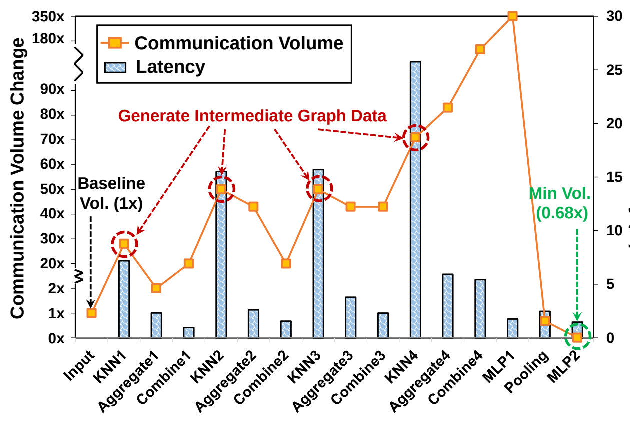


Fig. 2. Changes in the communication volume, transferred between operations, and the percentage of total latency for each operation in DGCNN on Jetson TX2.

design [21]. Most early GNN NAS research focused on enhancing model expressiveness [22], neglecting inference efficiency. To meet real-time edge application needs, hardware-aware GNN NAS has emerged to optimize both accuracy and efficiency. MaGNAS [18] presents a hardware-aware GNN NAS framework that uses a lookup table (LUT) to guide GNN search on the MPSoC platform. HGNAS [11] integrates a GNN-based hardware performance predictor into its NAS framework, achieving SOTA performance on various edge devices. However, these existing GNN NAS frameworks focus solely on single-device inference and do not effectively handle heterogeneous device-edge hierarchies in wireless network environments. GCoDE's system performance awareness is explored from a new perspective by abstracting the co-inference process into a unified GNN design space that naturally covers heterogeneous hardware sensitivities and network conditions.

3 GCoDE: Motivation & Overview

3.1 Why GNN Co-Inference is Inefficient

Below, We analyze the inefficiency bottlenecks in GNN co-inference through three observations using DGCNN [8] as the examined GNN model.

Observation ❶: The trade-off between computation and communication is difficult to manage. As shown in Fig. 2, the communication volume refers to the size of intermediate data—such as node features and, in some cases, graph structures—transferred between operations. For instance, the *KNN* operation can produce graph-structured intermediate data, significantly increasing the communication overhead when split after it. Additionally, it can be observed that as the feature dimension expands with more GNN layers, the *KNN* operation becomes increasingly time-consuming. Moreover, the split point that requires minimal intermediate data transmission is at the last *MLP*, which assigns most of the computation to the edge device. Since edge devices generally have much lower computational capability and fewer resources than edge servers, excessive computation on the device, especially for weaker platforms such as the Raspberry Pi, can cause severe on-device workload. This situation leads to underutilization of

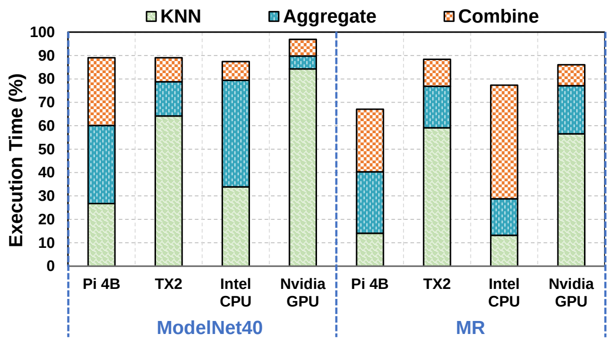


Fig. 3. Execution time breakdown of DGCNN across various devices and datasets.

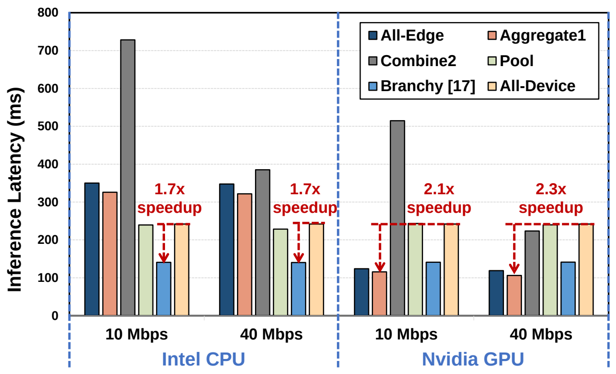


Fig. 4. Performance of various potential splitting schemes on DGCNN, with Jetson TX2 serving as the device.

edge resources and ultimately degrades the overall inference performance. Thus, designing an effective device-edge co-inference solution for GNNs suffers from a difficult trade-off between communication and computation.

Observation ❷: The hardware sensitivity of GNNs varies across heterogeneous hierarchies. Fig. 3 shows the varying hardware sensitivities of GNN operations, highlighting the need for heterogeneity awareness in design. For the point cloud dataset ModelNet40 [23], *KNN* operation is the main execution bottleneck on the Jetson TX2 and Nvidia GPU. This is because the parallel processing capability of these devices is hindered by the intensive and irregular memory accesses of the *KNN* operation. Additionally, the *Aggregate* operation becomes the main bottleneck on the Intel CPU, while on the Raspberry Pi, all operations are time-consuming due to resource constraints. Unlike point cloud data, the text dataset MR [24] features fewer nodes (1024 vs. 17) and larger feature dimensions (3 vs. 300), resulting in different execution characteristics. For example, the *Combine* operation became a computational bottleneck on the Intel CPU. This analysis shows that each GNN operation suits different platforms, explaining why traditional layer-level partitioning methods are inefficient.

Observation ❸: Detachment between architecture design and mapping design hinders system performance. Most traditional methods manually select the optimal split point within existing architectures to deploy GNNs to device-edge co-inference systems, such as Branchy-GNN [17]. Fig. 4 shows the performance of deploying DGCNN on device-edge systems based on various potential collaborative schemes. Considering that the NVIDIA 1060 GPU offers roughly 3× the computational capability of the Jetson TX2, their collaborative execution could intuitively achieve around a 4× speedup even without further optimizations. This indicates that the 2.3× improvement achieved by the prior method does not fully exploit the potential of collaboration and still leaves room for further optimization. The key issue of these inefficient results lies in the lack of co-optimization between the GNN architecture and the operation mapping scheme.

3.2 Why GCoDE Boosts System Efficiency

Fig. 5 shows an overview of the proposed GCoDE framework, which cleverly resolves the above dilemma. We first leverage a unified GNN design space construction method tailored for the co-inference paradigm to perfectly integrate the mapping scheme and GNN architecture into the same design space. Then, an efficient NAS approach is utilized to co-optimize the GNN architecture along with its embedded mapping scheme within this specialized design space for multiple objectives (e.g., accuracy, latency, and energy). The performance of candidate architectures during the exploration process is evaluated using system performance awareness methods, guiding the search toward more efficient solutions. Finally, leveraging the integrated GNN-specific deployment approach, efficient GNN co-inference is achieved. The motivating intuition is very simple: The unique aspect of device-edge co-inference is the communication process, and if we abstract this communication process as a special GNN operation, the sampled architecture from such a unified design space will inherently embed the mapping scheme. Consequently, this system-aware exploration of the unified design space can largely resolve the aforementioned dilemma. In this way, the designed collaborative solution evolves from a simple model split point to a novel GNN architecture with fine-grained operation mapping. Ultimately, GCoDE achieves optimal alignment among the system, GNN architecture, and mapping, significantly enhancing the efficiency.

4 Architecture-Mapping Co-Exploration

4.1 Problem Formulation

Leveraging our elegant abstraction of the co-inference process, collaborative schemes are inherently present in GNN architectures. Thus, co-exploring architectures and mappings reduces to exploring specialized GNN architectures with embedded mappings. Specifically, this work aims to co-optimize the accuracy and efficiency of GNNs deployed on device-edge co-inference systems. Given user requirements: edge device $\mathcal{D}$, edge server $\mathcal{E}$, network speed $\mathcal{S}$, latency constraint $\mathcal{C}_{lat}$ and on-device energy constraint $\mathcal{C}_e$, the optimization process can be formulated as:

$$\begin{aligned} &\arg\max_{\alpha\in\mathbb{A}} \left(acc_{val}\left(\mathcal{W}^*,\alpha\right) - \lambda\mathcal{P}_{sys}(\alpha,\mathcal{D},\mathcal{E},\mathcal{S})\right), \\ &s.t. \quad \mathcal{W}^* = \arg\max_{\mathcal{W}} \{acc_{train}\left(\mathcal{W},\alpha\right)\} \\ &\qquad \mathcal{L}_{sys} < \mathcal{C}_{lat} \quad and \quad E_{dev} < \mathcal{C}_e \end{aligned},$$

where $\mathcal{W}$ denotes the model weights, acc_{train} represents the training accuracy, acc_{val} is the validation accuracy, α refers to the architecture selected for optimization from the unified

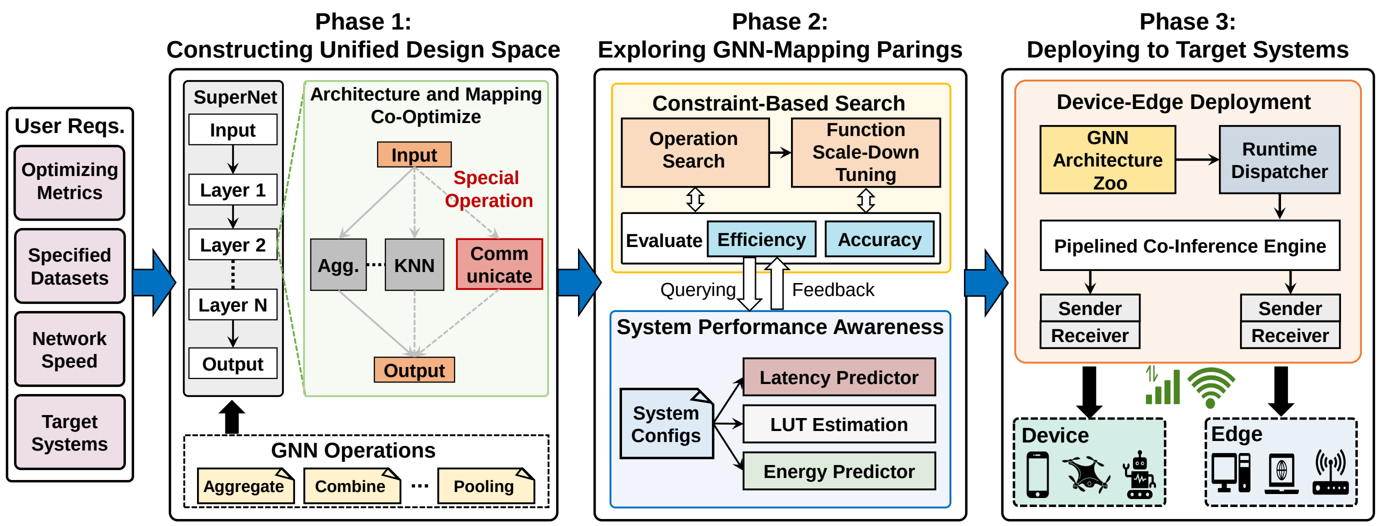
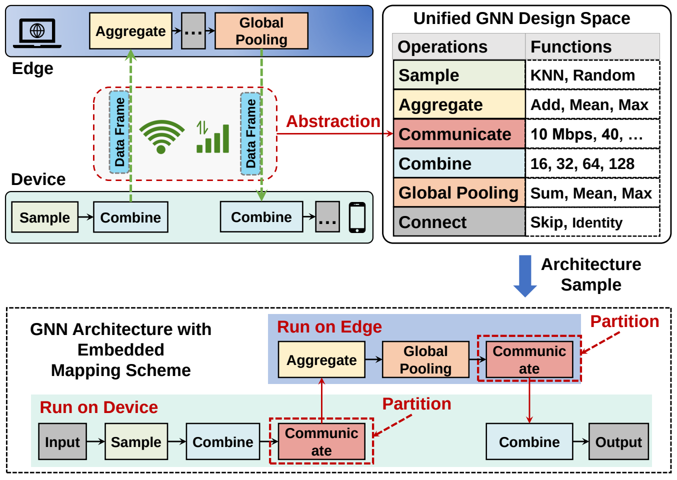


Fig. 5. Overview of GCoDE framework.



Unified GNN Design Space

Operations	Functions
Sample	KNN, Random
Aggregate	Add, Mean, Max
Communicate	10 Mbps, 40, …
Combine	16, 32, 64, 128
Global Pooling	Sum, Mean, Max
Connect	Skip, Identity

Fig. 6. Unified GNN design space for architecture design and operation mapping in device-edge co-inference.

GNN design space $\mathbb{A}$, $\mathcal{P}_{sys}$ indicates system performance, including inference latency $\mathcal{L}_{sys}$ and on-device energy consumption E_{dev}, and λ is a scaling factor that balances the trade-off between accuracy and efficiency. Note that system performance is jointly determined by the architecture, device-edge configurations, and network conditions.

4.2 Unified GNN Design Space for Co-Inference

Fig. 6 illustrates the unified design space for GNN device-edge co-inference, which supports the joint optimization of architectures and operation mappings. This is based on a novel concept: communication between the device and the edge can be treated as a specialized operation within the GNN architecture. As observed, the main difference between co-inference and on-device inference is that GNN operations are executed on different devices, using network communication to transfer intermediate data. Thus, we abstract *communicate* as a specialized GNN operation, constructing a unified design space for co-inference. In this way, GNN architectures sampled from this design space inherently incorporate operation mapping schemes across device-edge hierarchies. Consequently, joint optimization of architectures and collaborative schemes reduces to optimizing GNN architectures within the unified design space. Each *communicate* operation in the sampled GNN architecture represents a partition point, where the next operation is executed on the opposite side of the device-edge hierarchy. By integrating architecture and mapping design, GCoDE can explore more flexible collaboration patterns instead of merely seeking a split point. Furthermore, this fine-grained operation mapping enables the system to leverage device heterogeneity, maximizing each device's capabilities.

To utilize the one-shot NAS approach and avoid the overhead of retraining sub-architectures, GCoDE structures the unified design space $\mathbb{A}$ as a supernet, comprising six operations per layer: *Sample*, *Aggregate*, *Communicate*, *Combine*, *Global Pooling*, and *Connect*, each having distinct functional properties, as shown in Fig. 6. During supernet training, linear layers are employed to align the dimensions of all operations within each layer, and these layers are removed before the search to maintain efficiency.

4.3 Constraint-Based Search Strategy

While the unified design space offers benefits, it also introduces complexity to the exploration process. Each architecture sampled from this design space may include invalid configurations, such as consecutive *communicate* operations that are redundant and introduce unnecessary communication overhead. In such situations, intelligent algorithms, such as evolutionary algorithms (EA), are likely to struggle with identifying valid architectures instead of pursuing more efficient ones (see Sec. 7.6). When dealing with such complex design space, simpler random search strategies may lead to surprising outcomes and offer the potential for better optimization in a shorter duration [25]. Additionally, random search offers greater customizability and flexibility, enabling the discovery of multiple optimal solutions for different objectives at minimal cost. Thus, GCoDE integrates a constraint-based random search strategy to boost exploration efficiency, as depicted in Alg. 1.

Specifically, the search process consists of two stages: operation search and function scale-down tuning. Given

Algorithm 1: Constraint-based search strategy.

```
Input: Target device-edge co-inference system $Sys$,
       system constraints: $\{\mathcal{C}_{lat}, \mathcal{C}_e\}$, predefined
       function setting $f$, max operation search and
       function tuning iteration: $T$, $T_f$.
Output: The best found GNN architectures $\alpha^*$.
1  Initialize $\alpha^* \leftarrow \emptyset$, $Ops \leftarrow \emptyset$, $func \leftarrow \emptyset$.
2  Pre-train GNN supernet $\mathcal{N}_{super}$ with $f$.
3  /* Stage 1: operation search */
4  for $1 \leq t \leq T$ do
5      while Check($Ops$) do
6          $Ops \leftarrow$ Random ($\mathbb{A}$)  // Sample valid operations
7      end
8      $\alpha = \mathcal{N}_{super}(Ops, f)$          // Construct architecture
9      $\mathcal{P}_{sys} \leftarrow$ Evaluate ($Sys, \alpha$)  // Evaluate performance
10     if $\mathcal{L}_{sys} < \mathcal{C}_{lat}$ and $E_{dev} < \mathcal{C}_e$ then
11         $score \leftarrow (acc_{val} - \lambda \mathcal{P}_{sys})$       // Full evaluation
12     else
13         $score \leftarrow (-1)$      // Discard failed architectures
14     end
15     $\alpha^* \leftarrow$ Update($Ops, f, score$)  // Update operations
16 end
17 /* Stage 2: function scale-down tuning */
18 for $1 \leq t \leq T_f$ do
19     $f' \leftarrow$ Random($\mathbb{A}, f$)          // Scale-down functions
20     $\alpha^* \leftarrow$ Update($Ops, f', acc_{val}$)  // Update functions
21 end
22 return $\alpha^*$                         // Top-performing designs
```

the system configuration Sys and application requirements $\{\mathcal{C}_{lat}, \mathcal{C}_e\}$, GCoDE initially establishes an appropriate function setting f for the GNN supernet $\mathcal{N}_{super}$, referring to the target GNN model. Next, GCoDE trains the supernet, focusing on accuracy, to produce shared weights for further operation search. During the operation search, architecture validity is checked at each sampling step to avoid the overhead of evaluating invalid architectures. Subsequently, multi-objective optimization is conducted with a focus on achieving optimal system performance $\mathcal{P}_{sys}$ and validation accuracy acc_{val}. Finally, the function settings of the candidate architectures are further optimized during the function scale-down tuning stage to obtain the optimal GNN architecture. The details of the search process are outlined below.

Stage 1: Operation search. This stage aims to identify optimal operation settings Ops that allow candidate architecture α to achieve higher scores while meeting validity and performance constraints. Each candidate architecture is generated through random sampling of operations at each layer of the supernet. The validity of the candidate operation set is first evaluated using a checking function that identifies common invalid architecture configurations. System performance metrics, including latency $\mathcal{L}_{sys}$ and on-device energy consumption E_{dev}, are assessed only on valid architectures to ensure compliance with user requirements. In this manner, substandard candidates are quickly pruned, significantly improving exploration efficiency. The system performance of candidate architectures is evaluated using our unique performance-awareness method, reducing the overhead of real-time measurements. Subsequently, architectures meeting the criteria are evaluated for task accuracy using the shared weights from the pre-trained supernet, without any retraining. This follows the one-shot

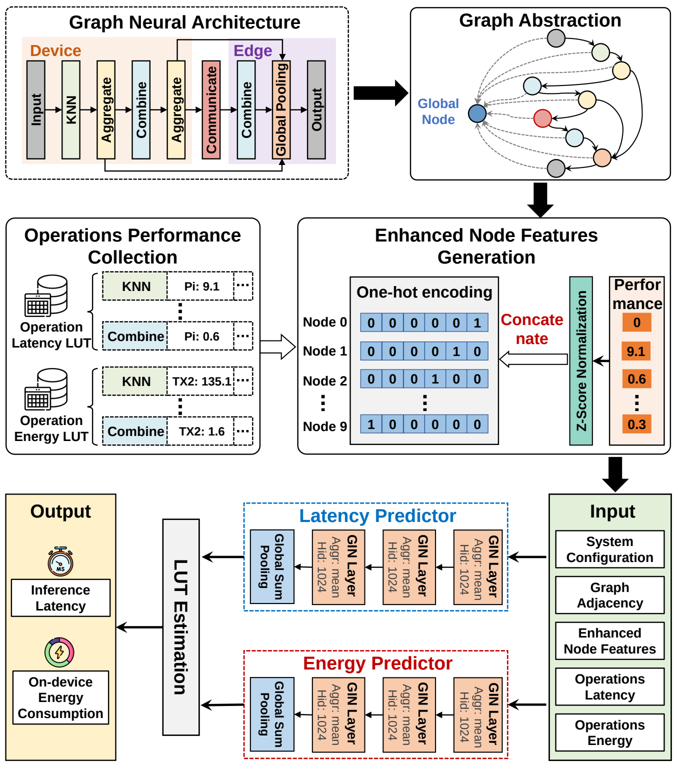


Fig. 7. Latency and on-device energy consumption prediction for GNNs on device-edge hierarchies.

NAS paradigm, which avoids retraining for each candidate and significantly improves the search efficiency. Finally, the optimal architectures α^* are updated by scoring the candidate operation set on accuracy and system performance metrics. Note that by leveraging the unified design space, the device–edge mapping is inherently embedded in each sampled architecture. This eliminates the need for a separate partitioning step and enables joint optimization of architecture and mapping in a single search process.

Stage 2: Function scale-down tuning. In this stage, GCoDE aims to identify function settings that maximize efficiency while maintaining acceptable accuracy loss. For example, by reducing feature dimensions in the *Combine* operations of candidate architectures to improve computational efficiency. As scaling down the function inevitably reduces computation, this phase focuses solely on evaluating the accuracy of the candidate architecture. Additionally, changes to function settings require retraining the sub-architecture, as supernet weights are not reusable, necessitating a balance between design duration and performance. Given that the candidate architecture from the operation search stage already satisfies requirements for accuracy, latency, and energy consumption, this stage is optional.

5 System Performance Awareness

To mitigate the considerable overhead from real-time measurements, GCoDE utilizes an efficient performance awareness method, as demonstrated in Fig. 7, to evaluate the performance of candidate GNN architectures deployed on the target device-edge co-inference system. Specifically, we focus on two crucial performance metrics in edge applications: inference latency and on-device energy consumption.

Additionally, the implementation of the proposed system performance awareness method, including training scripts, model architecture, and dataset preparation guidelines, is publicly available at https://github.com/BUAA-CI-LAB/GCoDE-Predictor.

5.1 Problem Simplification

The system performance awareness problem is greatly simplified by the unified GNN design space approach. Specifically, GNN architectures sampled from this space inherently contain implicit mapping schemes, heterogeneous device information, and network information. Thus, the challenge of assessing the performance of GNNs deployed on a heterogeneous device-edge co-inference system can be reframed as the task of extracting information from the GNN architecture. By abstracting the GNN architecture into an architecture graph, the problem is ultimately simplified to learning this architecture graph. Since GNNs excel at handling such graph-related problems, we can use a GNN-based predictor to learn system performance information from architecture graphs. Therefore, inspired by HGNAS [11], GCoDE integrates GNN predictors to assess the performance of GNN architectures on target device-edge co-inference systems.

5.2 Architecture Graph Construction

Unlike the performance awareness of GNNs on a single device, predictor learning faces more challenges in heterogeneous device-edge co-inference systems. To achieve accurate awareness across various system configurations, GCoDE introduces an enhanced node feature generation method. The architecture graph abstraction and the enhanced node feature generation are detailed as follows.

Graph abstraction. For a GNN architecture sampled from the unified design space, GCoDE first abstracts it into a directed acyclic graph. In this graph, nodes represent various operations, while edges indicate data flow between operations. To further enhance the connectivity of the architecture graph, GCoDE introduces global nodes to connect all other nodes and adds self-connections.

Enhanced node features generation. For GNN-based predictors, node features construction of the architecture graph commonly uses a one-hot encoding approach, as seen in HGNAS [11]. However, unlike the single-device focus of previous work, the device-edge hierarchies are generally heterogeneous. This indicates that identical operation in the same architecture graph might show vastly different execution characteristics. Therefore, the straightforward application of one-hot encoding strategies fails to effectively incorporate heterogeneous information, potentially resulting in poor performance of GNN-based predictors (See Sec. 7.5). To better capture system heterogeneity and network conditions, GCoDE combines lookup table (LUT) and predictor methods to construct enhanced node features for the architecture graph. Specifically, GCoDE maintains an LUT that records the performance of operations on different devices, with the overhead of constructing this lookup table being essentially negligible due to the limited number of valid operations. Subsequently, GCoDE concatenates the one-hot encoding of each node's type with its corresponding performance from the LUT, enhancing the information provided by the initial node features. As a result, the node features will comprise two components: operation type features and operation performance features. To mitigate the impact of different operation magnitudes, performance values in the LUT are normalized using *z-score normalization* before concatenation. In this manner, the initial node features in the architecture graph incorporate vital information about heterogeneous performance and network conditions, which support the effective learning of the GNN predictor.

5.3 Latency Prediction

With the system configuration, architecture graph, enhanced node features, and maintained operation latency LUT, GCoDE develops a GNN-based predictor model to estimate the architecture's latency on the target device-edge system. Since the architecture graph constructed by GCoDE contains sufficient system performance information, latency estimation can be considered a process of extracting information from the graph. As such, GCoDE builds the latency predictor based on three GIN layers. Additionally, each GIN layer uses a *mean* aggregation operator to aggregate latency information across nodes. Furthermore, *Global Sum Pooling* is applied at the tail of the predictor to further capture latency information aggregated by the GIN layers. The combination of the *mean* aggregation operator and *Global Sum Pooling* allows the predictor to efficiently extract latency information from the entire graph. Consequently, our latency predictor achieves high prediction accuracy, ensuring that the explored architectures meet the strict latency requirements of edge application scenarios. Additionally, since the architecture graph contains very few nodes, the predictor's runtime overhead is in the millisecond range, making it negligible.

In practice, incorrectly predicting high-latency architectures as low-latency can lead to substandard design. To avoid such mispredictions, GCoDE employs a LUT estimation approach to correct the predictor outputs. Specifically, using the latency LUT maintained by GCoDE, we estimate the architecture latency by simply accumulating all the operation latency in the corresponding architecture graph. Although it may not account for all potential runtime overheads, the estimated latency provides a lower bound. Thus, if the predicted result is lower than the estimated result, it indicates an incorrect prediction. In such cases, GCoDE corrects the prediction by adopting the LUT estimation as the evaluation result.

5.4 On-Device Energy Consumption Prediction

In real-world edge applications, the energy consumption on devices during co-inference is a crucial performance metric. Particularly for mobile devices, on-device energy consumption during inference determines the feasibility of the designed solution. To achieve the energy awareness, most existing work relies on traditional estimation methods [26], the same ones applied in our previous conference publications. Note that for device-edge co-inference, we focus primarily on the energy consumption of the device, as its energy is often limited. In this estimation approach,

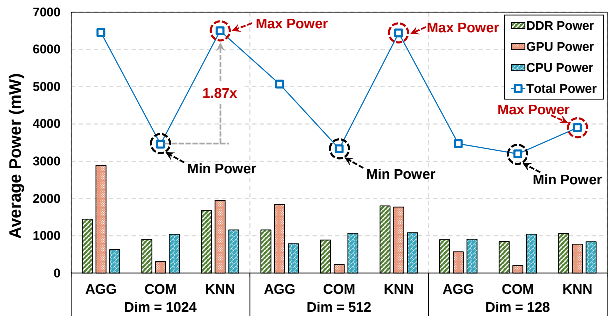


Fig. 8. Execution power of GNN operations on Jetson TX2.

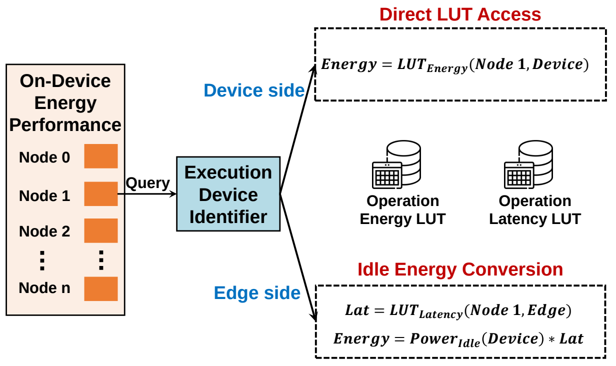

Direct LUT Access
Device side
$Energy = LUT_{Energy}(Node\ 1, Device)$
On-Device Energy Performance
Node 0
Node 1
Node 2
Node n
Query
Execution Device Identifier
Operation Energy LUT
Operation Latency LUT
Idle Energy Conversion
Edge side
$Lat = LUT_{Latency}(Node\ 1, Edge)$
$Energy = Power_{Idle}(Device) * Lat$

Fig. 9. Specialized performance feature construction for each node in energy predictor.

the total energy consumption of the device for a single co-inference process can be calculate as:

$$E_{total} = E_{idle} + E_{run} + E_{comm} \ ,$$

where E_{comm} represents the communication energy consumption, computed using power models proposed in [27]. E_{idle} and E_{run} are computational energy consumption in idle and operation executed states, respectively, calculated by multiplying associated power consumption with idle and execution time.

Those estimation methods assumes that the device's execution power remains constant throughout the co-inference process. However, as demonstrated in Observation 2, the execution characteristics of GNNs differ across various operations. This raises the question: Is it appropriate to use a fixed runtime power to estimate the energy consumption of various GNN architectures? Fig. 8 illustrates the profiling results of execution power for different GNN operations. It is clear that there are significant differences in the average power during the execution of various operations. For large feature dimensions, the *KNN* operation requires $1.87\times$ more average power than the *Combine* operation. This is due to the significantly increased DDR and GPU load resulting from the intensive memory accesses required by *KNN* operations. Additionally, the difference in execution power consumption between operations becomes more pronounced as the feature size increases. Moreover, the design solutions within the unified GNN design space may encompass a variety of different operation settings. Thus, using a fixed runtime power multiplied by execution time to calculate the energy consumption for each sampled GNN architecture is not reasonable. A fine-grained on-device energy awareness method is warranted to accurately consider the energy consumption differences among GNN operations.

To meet energy consumption requirements, GCoDE designs an on-device energy predictor to evaluate the energy usage of edge devices by candidate architectures during co-inference. We randomly sampled 9000 GNN architectures to gather on-device average power and latency data during co-inference, constructing the training dataset for the energy predictor. For the measurement of the average execution power on Jetson TX2, we leveraged the sensing circuitry integrated on the device. The energy consumption of each architecture is calculated by multiplying its latency by its average power. Additionally, we construct an operation energy LUT to record the execution energy consumption for various GNN operations. Following the construction of the training dataset, the energy predictor is trained for 500 epochs, utilizing *mean absolute percentage error* (MAPE) as the loss function.

As an extension of the system-aware approach, the model structure of the energy predictor is identical to that of the latency predictor. However, unlike the latency predictor, the energy consumption predictor requires a specialized design for constructing enhanced node features. This is because not all operations are executed on the device during co-inference. Therefore, directly querying the LUT for performance features, as done in the latency predictor, is not applicable to the energy consumption predictor. Additionally, simply summing the energy consumption of all operations fails to reflect the lower bound of on-device energy consumption. To this end, we developed a specialized method for constructing performance features for the energy predictor, as illustrated in Fig. 9. Specifically, when generating the performance feature part for each node, we first identified which side the node was executing on. For operation nodes executed on the device side, features were obtained directly by querying the operation energy LUT. While for the operation nodes executed on the edge side, we converted their energy consumption. The energy consumption of these nodes is estimated by multiplying the device's idle power consumption by the node's execution time. In this manner, the energy predictor effectively evaluates on-device energy consumption during GNN co-inference, enabling GCoDE to explore the most energy-efficient designs and meet energy consumption constraints.

6 Device-Edge Deployment

After identifying optimal GNN-Mapping pairings, GCoDE utilizes an integrated device-edge deployment approach to efficiently execute co-inference tasks on target systems. The three key components are the GNN architecture zoo, the runtime dispatcher, and the pipelined co-inference engine.

GNN architecture zoo. To address changing edge requirements, GCoDE maintains a collection of optimal architectures. This is achieved by leveraging the integrated constraint-based random search strategy, enabling simultaneous exploration of multiple optimization objectives within a single search process. Thus, we can identify the optimal architectures for varying requirements and system configurations in a single search process without extra overhead. In this way, GCoDE can quickly adapt its deployment

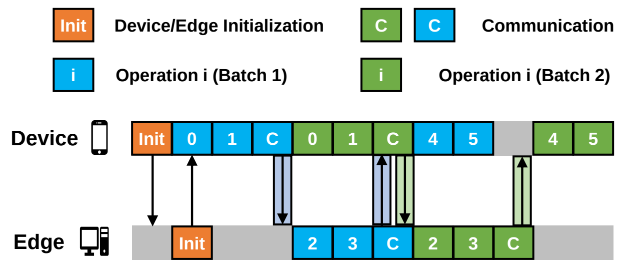


Fig. 10. Device-edge co-inference pipeline of two batches, with numbers in cells indicating operation IDs.

architecture to ensure efficient inference under varying system environments.

Runtime dispatcher. Leveraging the maintained GNN architecture zoo, GCoDE allows for rapid switching of deployment solutions during runtime. Specifically, GCoDE can choose the optimal GNN architecture for co-inference deployment according to user input or changes in the system environment. To allow flexibility in switching deployment architectures, GCoDE does not pre-assign GNN architectures to the devices and the edge. During each inference task execution, the edge device transmits the deployment architecture information to the edge server. As a result, the deployment architecture can be adjusted dynamically in response to the demands of edge devices.

Pipelined co-inference engine. There is a noticeable gap in operation-level fine-grained co-inference engines for GNNs, as most existing work is restricted to inter-layer collaboration. For this purpose, GCoDE develops a GNN-specific pipelined co-inference engine leveraging Python Socket [28]. Fig. 10 shows the device-edge co-inference pipeline of two batches, effectively boosting system throughput. Specifically, the device continues processing the next batch of data without idling while waiting for the edge to return intermediate results after executing the device-side operations of the deployed architecture. When network conditions degrade, the overlapping computation time can adequately cover communication delays, enhancing overall efficiency. Additionally, the co-inference engine utilizes multi-threading, with the sender and receiver operating independently, each maintaining its message queue. To minimize communication overhead, GCoDE serializes and compresses messages during each communication process. Moreover, GCoDE exhibits strong adaptability to heterogeneous environments owing to its modular communication design. In the prototype implementation, Python Socket is employed for rapid development and cross-platform validation, while the communication module can be readily replaced with alternative communication libraries to accommodate diverse deployment scenarios.

7 Experiment

7.1 Experimental Settings

Datasets and competitors settings. Our evaluation considers **two different application datasets**: the point cloud processing dataset ModelNet40 [23] and the text processing dataset MR [24]. To evaluate GCoDE's performance in device-edge co-inference, we compare it against **five baselines**: (1) the manually designed DGCNN model [8], (2) a manually optimized GNN architecture derived from DGCNN [10], (3) Branchy [17], a GNN device-edge co-inference approach leveraging model splitting, (4) PAS [29], a GNN NAS framework focused on graph classification tasks, and (5) HGNAS [11], a hardware-aware GNN NAS framework for edge devices. Additionally, to demonstrate that GCoDE's improvement in inference efficiency is not dependent on the computational power of the edge server, we established **three inference modes** for different competitors. Specifically, Device-Only (D) denotes that the inference task is performed entirely on the edge devices, Edge-Only (E) denotes that the data is sent to edge servers for inference, and Co-Inference (Co) denotes that the GNN inference task is distributed and executed across the device-edge hierarchy. Both GCoDE and Branchy, as co-inference methods, are executed in Co-Inference mode. Furthermore, we partitioned hardware-efficient GNNs from existing NAS frameworks and evaluated their efficiency at optimal split points to highlight the limitations of separating architecture and mapping design. For hyperparameter settings, we use the same setup as HGNAS for point cloud experiments and follow PAS for text processing. For a fair comparison, we used the reported task accuracy in these papers and evaluated efficiency based on the PyTorch Geometric (PyG) framework [30] under the same experimental conditions.

Device-edge system configurations. To compare the efficiency of GCoDE and competitors, we employ **four device-edge configurations**: Jetson TX2 [31] and Raspberry Pi 4B [32] as the device, and Nvidia 1060 GPU [33] and Intel i7-7700 CPU [34] as the edge. All devices are connected to a wireless router, with varying network conditions simulated by setting upload bandwidth limits ($\mathcal{S}_L$) to 10 Mbps and 40 Mbps. Furthermore, to ensure a fair comparison, all competitors use the same system configuration and co-inference engine as GCoDE in their experiments. Additionally, all implementations and tests are conducted based on the PyG framework, and the reported system performance results are from actual measurements on the target systems. The on-device energy consumption of Jetson TX2 is measured using its integrated sensing circuits, while the Raspberry Pi estimates running energy consumption following [26] since it lacks integrated sensing circuits.

Implementation settings for GCoDE. For the unified GNN design space, GCoDE built a 12-layer supernet and selected the functions from DGCNN as the initial function setting. During the GNN-mapping co-exploration, the operation search runs for up to 1000 iterations. Given the search efficiency and the architecture after the operational search were adequate for real-time applications, the function scale-down tuning process was omitted. For the latency and energy consumption predictors, we collected performance data from 9000 randomly sampled GNN architectures within the unified GNN design space to construct the training dataset (70%/30% for training/validation). Additionally, *mean absolute percentage error* (MAPE) is used as the loss function for training the predictor.

7.2 Evaluation on Point Cloud Processing

Overall performance. Fig. 11 illustrates the speedup comparison of GCoDE against all other SOTA baselines. GCoDE

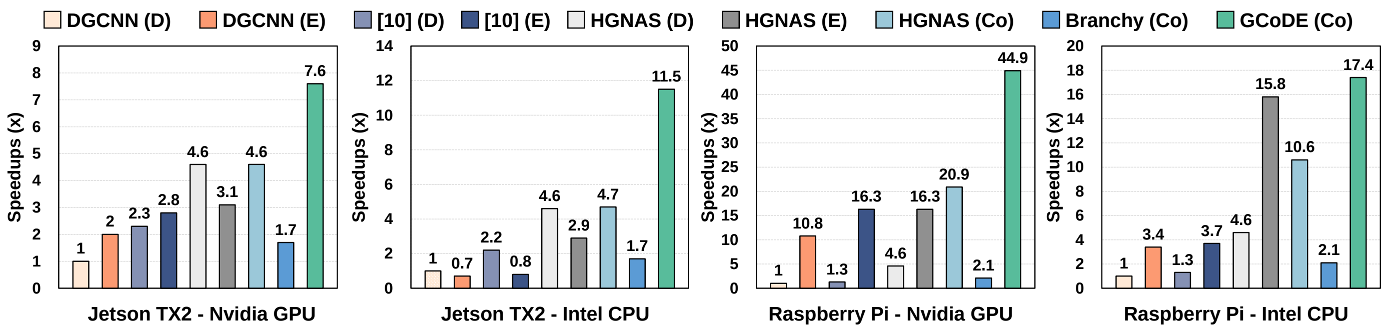


Fig. 11. The inference speedups achieved by our GCoDE framework over eight competitors, using DGCNN as the baseline.

TABLE 2
Performance comparison of GCoDE and existing approaches in different modes: Device-Only (D), Edge-Only (E), and Device-Edge Co-Inference (Co). OA and mAcc denote overall and balanced accuracy, respectively. Lat. and En. denote latency and on-device energy, respectively.

Edge	Mode	Method	OA	mAcc	Device: Jetson TX2				Device: Raspberry Pi 4B			
					$\mathcal{S}_L \leq$ 40 Mbps		$\mathcal{S}_L \leq$ 10 Mbps		$\mathcal{S}_L \leq$ 40 Mbps		$\mathcal{S}_L \leq$ 10 Mbps	
					Lat. (ms)	En. (J)	Lat. (ms)	En. (J)	Lat. (ms)	En. (J)	Lat. (ms)	En. (J)
-	D	DGCNN [8]	92.9	88.9	241.9	1.0	241.9	1.0	1121.8	5.6	1121.8	5.6
		[10]	92.6	90.6	107.6	0.4	107.6	0.4	851.1	4.3	851.1	4.3
		HGNAS [11]	92.1∼92.2	88.3∼88.7	52.1	0.2	52.1	0.2	241.5	1.2	241.5	1.2
Nvidia GPU	E	DGCNN [8]	92.9	88.9	118.8	0.2	123.9	0.3	103.5	0.4	107.8	0.4
		[10]	92.6	90.6	86.6	0.2	93.4	0.2	68.7	0.2	75.8	0.3
		HGNAS [11]	92.1	88.5	79.2	0.2	87.8	0.2	69.0	0.2	70.3	0.3
	Co	Branchy [17]	92	-	141.2	0.6	141	0.6	541.8	2.6	531.8	2.6
		HGNAS [11]	92.1∼92.2	88.3∼88.7	52.6	0.2	57.1	0.2	53.7	1.9	72.9	0.9
		GCoDE	**92.1∼92.8**	**88.1∼89.7**	**31.9**	**0.1**	**39**	**0.1**	**25.0**	**0.1**	**35.6**	**0.1**
Intel CPU	E	DGCNN [8]	92.9	88.9	347.6	0.8	350.1	0.8	333.7	1.0	339.5	1.1
		[10]	92.6	90.6	321.6	0.7	325.5	0.8	303.4	1.0	307.7	1.0
		HGNAS [11]	92.1	88.5	83.7	0.2	88.3	0.2	71.0	0.3	74.0	0.3
	Co	Branchy [17]	92	-	140.2	0.6	140.8	0.6	528.1	2.5	544.0	2.6
		HGNAS [11]	92.1∼92.2	88.3∼88.7	51.4	0.2	53.8	0.2	106.0	2.1	122.8	1.0
		GCoDE	**92.0∼92.6**	**88.9∼89.4**	**21**	**0.1**	**50.2**	**0.2**	**64.4**	**0.2**	**49.3**	**0.2**

achieves the most significant inference efficiency improvements across all system configurations, with speedups of $7.6\times$, $11.5\times$, $44.9\times$, and $17.4\times$, respectively. The acceleration of GCoDE is even more significant with the Jetson TX2 - Intel CPU and Raspberry Pi - Nvidia GPU system configurations. This phenomenon aligns with the hardware sensitivities noted in our earlier observation (Observation 2). Given the differing execution bottlenecks of edge devices and edge servers in these two configurations, GCoDE effectively recognizes and utilizes this heterogeneity to achieve greater performance improvements.

GCoDE vs. Existing approaches. Tab. 2 provides detailed experimental results, including comparisons of accuracy, latency, and on-device energy consumption. Compared to directly deploying manually designed DGCNN models on target edge devices, GCoDE achieves up to $44.9\times$ speedups and 98.2% energy savings while maintaining similar accuracy. Against hardware-efficient GNNs designed by HGNAS for edge devices, GCoDE consistently shows optimal performance, achieving up to $9.7\times$ speedup and 91.6% energy savings on the lower-powered Raspberry Pi. The much higher energy reduction ratio compared to latency reduction is due to the large performance gap between the Raspberry Pi and the edge server, which leads the optimizer to offload most operations to the server. As our measured energy consumption only accounts for on-device energy, this results in a relatively greater energy reduction than latency reduction. Furthermore, compared to these efficient GNN architectures operating in Edge-Only mode, GCoDE still provides up to $5.7\times$ acceleration and over 50% energy savings. This demonstrates that GCoDE effectively leverages the potential of the device-edge co-inference paradigm to achieve optimal system performance. Compared to the GNN device-edge co-inference approach Branchy, GCoDE achieves up to $21.9\times$ speedups and 96.1% energy savings. Additionally, GCoDE outperforms Branchy across all tests, even under varying network conditions and device-edge hierarchies. This can be attributed to GCoDE's ability to jointly optimize the GNN architecture and operation mapping while leveraging system performance awareness to explore optimal designs. Compared to HGNAS with its best partitioning point selection, GCoDE still achieves up to $2.5\times$ inference speedups. This highlights that the separation of architecture design and operation mapping leads to sub-optimal performance, while GCoDE's co-optimization approach maximizes system performance.

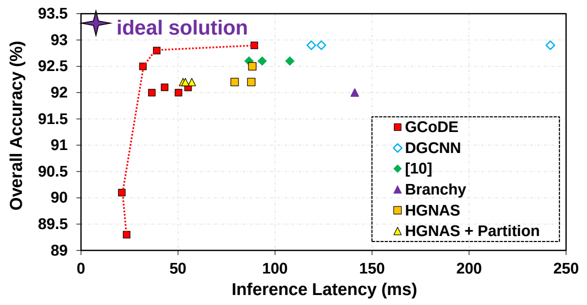


Fig. 12. Comparison between the existing approaches and GCoDE in terms of accuracy and latency.

TABLE 3
Comparison of existing methods and GCoDE on MR dataset.

		GCoDE	Branchy [17]	PAS [29]		PAS [29]
Accuracy (%)		**76.1∼77.0**	75.5	76.7		76.7
Mode		**Co**	Co	D	E	Co
TX2 ⇌ GPU	Lat. (ms)	**8.7**	26.4	29.1	30.7	16.2
	En. (mJ)	**25.3**	78.3	94.3	71.4	49.3
TX2 ⇌ CPU	Lat. (ms)	**8.5**	29.0	29.1	18.6	15.5
	En. (mJ)	**24.5**	87.9	94.3	45.4	47.2
Pi ⇌ GPU	Lat. (ms)	**4.8**	32.3	13.6	32.0	8.0
	En. (mJ)	**30.0**	150.0	70.0	140.0	41.0
Pi ⇌ CPU	Lat. (ms)	**2.00**	28.7	13.6	28.7	6.9
	En. (mJ)	**10.0**	140.0	70.0	110.0	37.0

Accuracy vs. efficiency. Fig. 12 illustrates the GNN design space exploration results using Jetson TX2 as the device. GCoDE advances the Pareto frontier in GNN inference performance beyond all baselines, achieving both higher accuracy and lower latency. This advancement is due to our system performance predictor, which identifies efficient GNN architectures, bringing GCoDE closer to the ideal solution. Furthermore, the selection of scaling factor λ in the search process allows users to tailor the GNN co-inference architecture for either higher accuracy (smaller λ) or lower latency (larger λ), based on their needs.

7.3 Evaluation on Text Processing

To evaluate the performance of GCoDE across different applications, we conducted experiments on the MR text analysis dataset under a 40 Mbps network condition. The experimental results are illustrated in Tab. 3, which demonstrate that GCoDE consistently maintains superior accuracy and system efficiency compared to all competitors. Specifically, GCoDE outperforms PAS, which is deployed in Device-Only mode, achieving speedups of 3.3×, 3.4×, 2.8×, and 6.8× across four heterogeneous system configurations. Compared to the architecture designed by PAS and deployed in Edge-Only mode, GCoDE achieves 3.5×, 2.2×, 6.7×, and 14.4× improvements in inference efficiency across four system configurations, respectively. Against co-inference methods like Branchy and PAS, which utilize optimal partitioning points, GCoDE achieves up to 14.3× speedup. Furthermore, GCoDE is the most energy-efficient approach among all baselines, requiring only 10 mJ on the Raspberry Pi for a single inference.

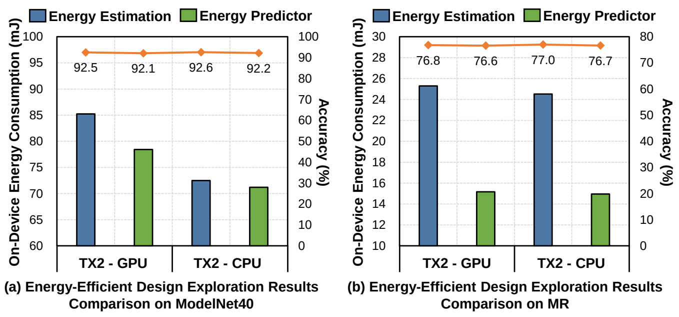


Fig. 13. Performance comparison of exploration based on energy predictor or energy estimation.

7.4 Energy-Efficient Design Exploration Results

Fig. 13 (a) shows the results of energy-efficient design exploration on the ModelNet40 dataset. During the exploration, the on-device energy consumption of the candidate architectures is evaluated using the traditional estimation method and our predictor, respectively. With a more accurate energy predictor, GCoDE effectively explores GNN architectures that consume less on-device energy during co-inference. Specifically, the prediction method achieves up to 8% reduction in energy consumption on point cloud processing. Furthermore, Fig. 13 (b) illustrates that with energy prediction methods, GCoDE achieves a further reduction in on-device energy consumption by 40% and 39% for text processing tasks, respectively. In practice, GCoDE-designed architectures aimed at faster inference often show lower on-device energy consumption, aligning with the findings of [35]. Thus, even though the energy estimation method is not accurate, architectures with relatively low energy consumption can still be identified by considering the impact of latency on the objective function. Nonetheless, the effectiveness of architectures based on energy estimation methods remains uncertain, making them unsuitable for scenarios with strict energy constraints. Conversely, the energy predictor provides outputs closer to the measured values, ensuring that the designed solution adheres to strict energy constraints.

7.5 System Performance Awareness Results

In this section, we evaluate the performance of GCoDE in predicting latency and on-device energy consumption across four different device-edge system configurations. Meanwhile, we extend the GNN hardware performance predictor proposed by HGNAS to enable awareness at the device-edge hierarchy, allowing for a fair comparison.

Latency prediction. Fig. 14 presents an intuitive illustration of the performance of our proposed latency predictor across different system configurations. It is clear that our prediction results are very close to the measured co-inference latency on the target device-edge systems. The average MAPE of latency predictions across the four device-edge hierarchies is approximately 6.9. Additionally, GCoDE demonstrates a notable improvement in accuracy for device-edge co-inference latency prediction over the HGNAS predictor, as illustrated in Fig. 15. Specifically, GCoDE achieves latency prediction accuracies of 75%, 72%, 85%, and 83% across various system configurations within a 10% error

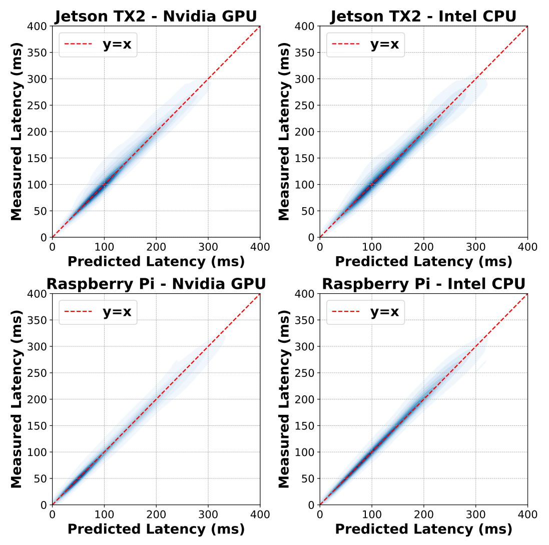


Fig. 14. Illustration of the relationships between predicted and measured latency across four device-edge systems.

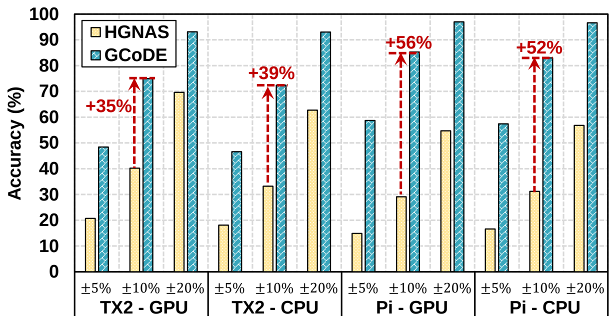


Fig. 15. Latency prediction accuracy within 20% error bound on four device-edge systems.

bound. Compared to HGNAS, GCoDE achieves a prediction accuracy improvement of up to 52% in GNN device-edge co-inference latency. This is because HGNAS relies on a simple one-hot encoding strategy, and the node features in the architecture graph lack essential heterogeneous performance information. In contrast, the enhanced node feature construction method introduced by GCoDE effectively incorporates system performance and heterogeneity information into the architecture graph, significantly facilitating the learning of the GNN predictor. Besides, Fig. 16 further shows that GCoDE accurately evaluates the relative latency among candidate architectures, surpassing 94.7% accuracy. Moreover, to further assess the robustness of the proposed performance predictor, we evaluated it under diverse dynamic conditions, including varying network bandwidths (1, 20, 40, and 100 Mbps) and different edge workloads. Using approximately 600 samples collected across these scenarios, the predictor achieved a relative latency ranking accuracy of 87.11%. This high latency-aware performance is due to GCoDE's elegant abstraction of co-inference, which transforms complex system awareness problems into graph learning problems, allowing GNN predictors to solve them. Furthermore, the latency prediction overhead is measured in milliseconds and is negligible. By leveraging the latency predictor, GCoDE can efficiently identify the optimal GNN

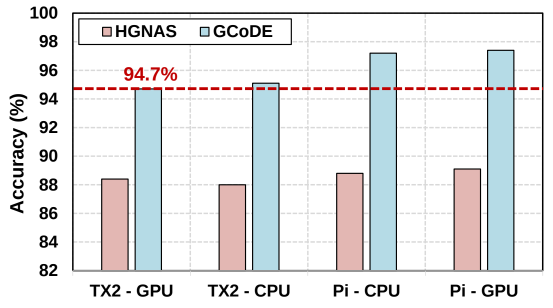


Fig. 16. Prediction accuracy of relative latency ranking among candidate architectures.

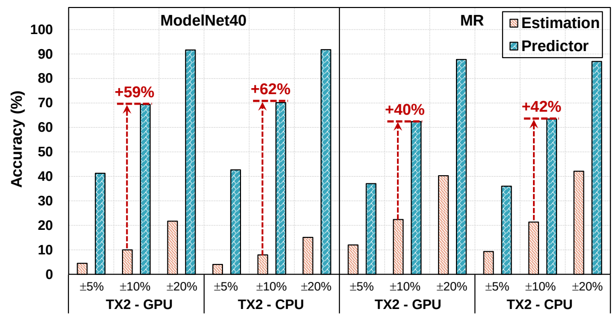


Fig. 17. On-device energy consumption prediction accuracy within 20% error bound on four device-edge systems.

architecture with a mapping scheme for the target system.

Energy prediction. Fig. 17 illustrates the accuracy of the energy predictor in GCoDE. Traditional estimation methods do not accurately reflect actual on-device energy consumption during co-inference, making it hard to meet energy constraints in practice. By extending the predictor's awareness dimension, GCoDE achieves high accuracy in energy prediction through a specialized method for constructing the performance parts of node features. Specifically, the energy predictor achieves prediction accuracies of 69%, 70%, 62%, and 63% within a 10% error bound across various system configurations and applications, respectively. Compared to estimation methods, GCoDE achieves up to 62% and 42% improvements in energy consumption prediction accuracy for different applications, respectively. Moreover, with a 20% error bound, GCoDE can achieve over 87% energy prediction accuracy. Additionally, our energy predictor achieves up to 95% accuracy in predicting relative latency among candidate architectures. By leveraging this accurate energy predictor, GCoDE can adhere to on-device energy consumption constraints of edge applications and identify more energy-efficient GNN solutions.

7.6 Ablation Studies

In this section, we evaluate the effectiveness of the proposed constraint-based random search strategy in enhancing search efficiency, as well as the performance of various predictor construction methods.

Random search vs. Evolutionary search. As shown in Fig. 18 (a), due to the presence of many invalid architectures in the unified GNN design space, the evolutionary search method gets stuck in a loop of identifying valid architectures, thereby losing the ability to search for higher-

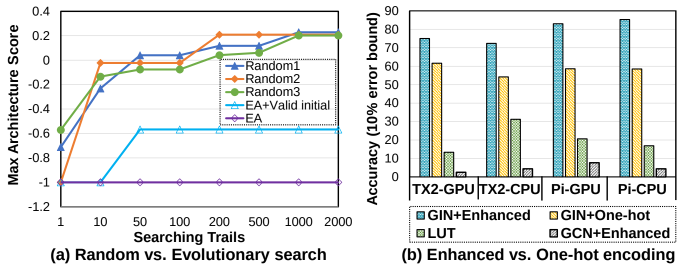


Fig. 18. (a) Efficiency comparison between random search and evolutionary search. (b) Prediction accuracy improvement with GCoDE method (GIN + Enhanced feature).

performing ones. In contrast, the proposed constraint-based random search strategy performs remarkably well and can identify the optimal designs in fewer trials. Specifically, GCoDE requires only 1.5 GPU hours to perform the search on the ModelNet40 dataset, doubling the exploration efficiency compared to the 3 GPU hours required by HGNAS. For MR datasets with smaller sizes, GCoDE further reduces the search time to 0.2 GPU hours, compared to PAS, which requires 0.27 GPU hours.

Enhanced feature construction vs. One-hot encoding strategy. Fig. 18 (b) shows a performance comparison of various prediction methods for device-edge co-inference. Although the LUT method can evaluate performance lower bounds for GNN architectures, it struggles to achieve high prediction accuracy across different system configurations. The reason is that it neglects essential runtime overhead. Furthermore, GIN demonstrates a clear advantage over GCN in learning architecture graphs, owing to its superior graph information learning capability. Moreover, while the powerful graph information extraction capability of GIN leads to an improvement in the accuracy of the one-hot encoding strategy, it remains ineffective. In contrast, the combination of our enhanced node feature construction method and GIN's powerful learning capability leads to a significant improvement in prediction accuracy across various heterogeneous co-inference systems.

8 Conclusions

In this paper, we propose GCoDE, the first system-aware automated framework for designing and deploying GNNs on device-edge co-inference systems. Given user requirements, GCoDE can automatically design the optimal GNN architecture with an embedded operation mapping scheme, while providing efficient co-inference engine support. GCoDE constructs a unified architecture-mapping co-design space, employing constraint-based search strategies and accurate system performance awareness approaches to identify optimal solutions. Extensive experiments demonstrate that GCoDE achieves superior accuracy, inference speed, and energy efficiency across diverse applications and systems, surpassing baselines with up to $44.9\times$ acceleration and 98.2% energy reduction. We believe that GCoDE brings significant heuristic advances in deploying efficient GNNs for large-scale wireless network edge applications.

References

[1] Y. Sheng, J. Yang *et al.*, "The larger the fairer? small neural networks can achieve fairness for edge devices," in *Proceedings of ACM/IEEE Design Automation Conference (DAC)*, 2022, pp. 163–168.

[2] Z. Xue, Y. Yang *et al.*, "Sugar: Efficient subgraph-level training via resource-aware graph partitioning," *IEEE Transactions on Computers (TC)*, 2023.

[3] B. Yan, C. Wang *et al.*, "TinyGNN: Learning efficient graph neural networks," in *Proceedings of ACM International Conference on Knowledge Discovery & Data Mining (KDD)*, 2020, pp. 1848–1856.

[4] S. A. Tailor, R. de Jong *et al.*, "Towards efficient point cloud graph neural networks through architectural simplification," in *Proceedings of IEEE/CVF International Conference on Computer Vision (ICCV)*, 2021, pp. 2095–2104.

[5] F. Chen, J. Shao *et al.*, "Multivariate, multi-frequency and multimodal: Rethinking graph neural networks for emotion recognition in conversation," in *Proceedings of IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, 2023.

[6] W. Shi and R. Rajkumar, "Point-GNN: Graph neural network for 3D object detection in a point cloud," in *Proceedings of the IEEE/CVF conference on computer vision and pattern recognition (CVPR)*, 2020, pp. 1711–1719.

[7] P. Dighe, S. Adya *et al.*, "Lattice-based improvements for voice triggering using graph neural networks," in *Proceddings of IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP)*, 2020, pp. 7459–7463.

[8] Y. Wang, Y. Sun *et al.*, "Dynamic graph CNN for learning on point clouds," *ACM Transactions on Graphics (TOG)*, vol. 38, no. 5, pp. 1–12, 2019.

[9] R. Kalliomäki, "Real-time object detection for autonomous vehicles using deep learning," 2019.

[10] Y. Li, H. Chen *et al.*, "Towards efficient graph convolutional networks for point cloud handling," in *Proceedings of IEEE/CVF International Conference on Computer Vision (ICCV)*, 2021, pp. 3752–3762.

[11] A. Zhou, J. Yang *et al.*, "Hardware-aware graph neural network automated design for edge computing platforms," in *Proceedings of ACM/IEEE Design Automation Conference (DAC)*, 2023.

[12] Q. Lu, W. Jiang *et al.*, "Hardware/Software co-exploration for graph neural architectures on FPGAs," in *Proceedings of IEEE Computer Society Annual Symposium on VLSI (ISVLSI)*, 2022, pp. 358–362.

[13] W.-Q. Ren, Y.-B. Qu *et al.*, "A survey on collaborative DNN inference for edge intelligence," *Machine Intelligence Research*, vol. 20, no. 3, pp. 370–395, 2023.

[14] M. S. Murshed, C. Murphy *et al.*, "Machine learning at the network edge: A survey," *ACM Computing Surveys (CSUR)*, vol. 54, no. 8, pp. 1–37, 2021.

[15] J. Li, G. Liao *et al.*, "Roulette: A semantic privacy-preserving device-edge collaborative inference framework for deep learning classification tasks," *IEEE Transactions on Mobile Computing (TMC)*, 2023.

[16] X. Li and S. Bi, "Optimal AI model splitting and resource allocation for device-edge co-inference in multi-user wireless sensing systems," *IEEE Transactions on Wireless Communications (TWC)*, 2024.

[17] J. Shao, H. Zhang *et al.*, "Branchy-GNN: A device-edge co-inference framework for efficient point cloud processing," in *Proceedings of IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP)*, 2021.

[18] M. Odema, H. Bouzidi *et al.*, "MaGNAS: A mapping-aware graph neural architecture search framework for heterogeneous MPSoC deployment," *ACM Transactions on Embedded Computing Systems (TECS)*, 2023.

[19] M. Yan, Z. Chen *et al.*, "Characterizing and understanding GCNs on GPU," *IEEE Computer Architecture Letters (CAL)*, vol. 19, no. 1, pp. 22–25, 2020.

[20] A. Das, S. Kundu *et al.*, "Grannite: Enabling high-performance execution of graph neural networks on resource-constrained neural processing units," *arXiv preprint arXiv:2502.06921*, 2025.

[21] Y. Gao, H. Yang *et al.*, "Graph neural architecture search," in *Proceedings of International joint conference on artificial intelligence (IJCAI)*. International Joint Conference on Artificial Intelligence, 2021.

[22] S. Cai, L. Li *et al.*, "Rethinking graph neural architecture search from message-passing," in *Proceedings of the IEEE/CVF conference on computer vision and pattern recognition (CVPR)*, 2021, pp. 6657–

6666.

[23] Z. Wu, S. Song *et al.*, "3D ShapeNets: A deep representation for volumetric shapes," in *Proceedings of IEEE conference on computer vision and pattern recognition (CVPR)*, 2015.

[24] Y. Zhang, X. Yu *et al.*, "Every document owns its structure: Inductive text classification via graph neural networks," in *Proceedings of Annual Meeting of the Association for Computational Linguistics (ACL)*, 2020.

[25] K. Yu, C. Suito *et al.*, "Evaluating the search phase of neural architecture search," in *Proceedings of International Conference on Learning Representations (ICLR)*, 2020.

[26] M. Odema, N. Rashid *et al.*, "LENS: Layer distribution enabled neural architecture search in edge-cloud hierarchies," in *Proceedings of ACM/IEEE Design Automation Conference (DAC)*, 2021.

[27] J. Huang, F. Qian *et al.*, "A close examination of performance and power characteristics of 4G LTE networks," in *Proceedings of international conference on Mobile systems, applications, and services (MobiSys)*, 2012.

[28] "Python socket," [Online]. Available: https://docs.python.org/3/library/socket.html.

[29] L. Wei, H. Zhao *et al.*, "Neural architecture search for GNN-based graph classification," *ACM Transactions on Information Systems (TOIS)*, 2023.

[30] M. Fey and J. E. Lenssen, "Fast graph representation learning with PyTorch Geometric," in *Proceedings of International Conference on Learning Representations (ICLR)*, 2019.

[31] "NVIDIA. Jetson TX2," [Online]. Available: https://www.nvidia.com.

[32] "Raspberry Pi 4B," [Online]. Available: https://www.raspberrypi.com.

[33] "NVIDIA. GeForce GTX1060," [Online]. Available: https://www.nvidia.com.

[34] "Intel. Core i7-7700 Processor," [Online]. Available: https://www.intel.com.

[35] X. Luo, D. Liu *et al.*, "SurgeNAS: a comprehensive surgery on hardware-aware differentiable neural architecture search," *IEEE Transactions on Computers (TC)*, vol. 72, no. 4, pp. 1081–1094, 2022.

Ao Zhou received the B.S. and M.S. degrees in Software Engineering from Beijing University of Technology, Beijing, China, in 2018 and 2021, respectively, and the Ph.D. degree in Software Engineering from Beihang University, Beijing, China, in 2025. He is currently a Postdoctoral Research Fellow at Beihang University. His research interests include GNN acceleration, LLM inference systems, and heterogeneous computing. He is one of the contributors to the PyTorch Geometric (PyG).

Jianlei Yang (S'11-M'14-SM'20) received the B.S. degree in microelectronics from Xidian University, Xi'an, China, in 2009, and the Ph.D. degree in computer science and technology from Tsinghua University, Beijing, China, in 2014. He is currently a Professor in Beihang University, Beijing, China, with the School of Computer Science and Engineering. From 2014 to 2016, he was a post-doctoral researcher with the Department of ECE, University of Pittsburgh, Pennsylvania, USA. His current research interests include emerging computer architectures, hardware-software co-design, and machine learning systems. Dr. Yang was the recipient of the First/Second place on ACM TAU Power Grid Simulation Contest in 2011/2012. He was a recipient of IEEE ICCD Best Paper Award in 2013, ACM GLSVLSI Best Paper Nomination in 2015, IEEE ICESS Best Paper Award in 2017, ACM SIGKDD Best Student Paper Award in 2020.

Tong Qiao received the B.S. degree in computer science and technology from Beihang University, Beijing, China, in 2020. He is currently pursuing the Ph.D. degree at the School of Computer Science and Engineering, Beihang University, China. His research interests include graph neural networks acceleration, and system for machine learning.

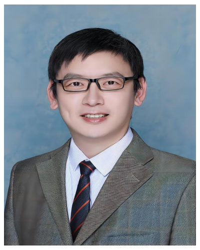
Yingjie Qi received the B.S. degree in computer science and technology from Beihang University, Beijing, China, in 2020. He is currently pursuing the Ph.D. degree at the School of Computer Science and Engineering, Beihang University, China. His research interests include compute-in-memory architectures, deep learning compilers, and graph neural networks acceleration.

Zhi Yang is currently an associate researcher in the School of Computer Science in Peking University. He obtained his Ph.D. from Peking University in 2010. His major research interests include AI computing systems and distributed systems. He was a recipient of VLDB Best Scalable Data Science Paper in 2022, WWW Best Student Paper Award in 2022.

Weisheng Zhao (Fellow, IEEE) received the Ph.D. degree in physics from the University of Paris Sud, Paris, France, in 2007.

He is currently a Professor with the School of Integrated Circuit Science and Engineering, Beihang University, Beijing, China. In 2009, he joined the French National Research Center, Paris, as a Tenured Research Scientist. Since 2014, he has been a Distinguished Professor with Beihang University. He has published more than 300 scientific articles in leading journals and conferences, such as *Nature Electronics*, *Nature Communications*, *Advanced Materials*, IEEE Transactions, ISCA, and DAC. His current research interests include the hybrid integration of nanodevices with CMOS circuit and new nonvolatile memory (40-nm technology node and below) like MRAM circuit and architecture design.

Prof. Zhao was the Editor-in-Chief for the IEEE TRANSACTIONS ON CIRCUITS AND SYSTEM I: REGULAR PAPER from 2020 to 2023.



Chunming Hu received the PhD degree in computer science and technology from Beihang University, Beijing, China, in 2006.

He is currently a professor and dean of the School of Software, Beihang University, Beijing, China. His current research interests include distributed systems, system virtualization, mobile computing and cloud computing.

Prof. Hu is currently one of the W3C Board of Directors, and serving as the Deputy Director of W3C China.