

HGNN-Part: A High-Quality Hypergraph Partitioner Based on Hypergraph Generative Model

Shengbo Tong, Rufan Zhou, Chunyan Pei, and Wenjian Yu

Department of Computer Science and Technology, BNRist, Tsinghua University, Beijing, China

Abstract—Hypergraph partitioning is a fundamental combinatorial optimization problem with critical applications in VLSI design. While recent deep learning based approaches have shown promise for this problem, they rely on graph neural networks (GNNs) that require transforming hypergraphs into normal graphs, thereby losing the high-order relationships in hypergraph structures. In this work, we propose a novel framework that directly utilizes hypergraph neural networks (HGNNs) to exploit the high-order interactions in hypergraphs. We develop an efficient normalized cut loss computation algorithm optimized for GPU training and apply randomized matrix decomposition techniques to significantly accelerate the eigenvector computation required for node feature extraction without sacrificing quality. To address the scarcity of open-source hypergraph data, we release a comprehensive dataset with 164 VLSI hypergraphs collected from various EDA contests and benchmarks. Extensive experiments on the ISPD98 and ISPD05 benchmarks demonstrate that our method achieves superior partitioning quality compared to state-of-the-art approaches, including multilevel methods (hMETIS), spectral methods (SpecPart, K-SpecPart), and recent deep learning based approaches (MedPart, GenPart). Furthermore, training on our expanded dataset yields additional performance gains, validating the framework’s ability to leverage larger training data effectively.

Index Terms—hypergraph partitioning, VLSI design

I. INTRODUCTION

Hypergraph partitioning is a critical combinatorial optimization problem with applications in various domains. Its goal is to partition a hypergraph into smaller and balanced subgraphs while minimizing the cutsize, which is the total weight of hyperedges that connect different partitions. Many hypergraph partitioning methods have been proposed, including multilevel recursive-bisection approaches [1]–[4], spectral methods [5], [6], and more recently, deep learning based techniques [7]–[9].

VLSI design is an important application area for hypergraph partitioning, as modern EDA tools use partitioning tools throughout the design flow [10]. Effective partitioning can significantly impact the performance, power consumption, and area of the final design. Applying multilevel partitioning techniques has already shown promising results in improving the quality of VLSI layouts [11], [12], but its main drawback is the lack of global information, which can lead to sub-optimal solutions. Spectral methods consume too much time and memory for large-scale problems because of the large amount of numerical computation. Therefore, deep learning based approaches have now emerged as a promising alternative, leveraging the fast computing speed of GPUs and the power of neural networks to learn effective partitioning strategies.

However, although partitioners based on deep learning have shown promise for general hypergraphs, directly using them on VLSI benchmarks often leads to unsatisfactory results. One reason is that graph neural networks (GNNs) [13] are still the main solution for introducing machine learning techniques on this problem, but applying GNNs demands transforming hypergraphs into normal graphs, which may not fully capture the high-order relations in hypergraphs [10]. Another reason is that the structure of VLSI hypergraphs is significantly different from that of general hypergraphs, reflecting the unique characteristics and complex interconnections present in VLSI designs [10].

GenPart [9] is the first study that has demonstrated the feasibility of applying deep learning to hypergraph partitioning in the context of VLSI design. It solves the second problem mentioned above by combining deep learning methods with traditional multilevel techniques, allowing for more sophisticated adjustments during the optimization process. However, it still relies on GNNs and does not fully leverage the high-order relationships in hypergraphs. Moreover, it only uses the ISPD98 benchmarks for training, which may not be sufficient to capture the diverse structures of VLSI hypergraphs.

To address these limitations, we propose a novel framework HGNN-Part¹, which utilizes hypergraph neural networks (HGNNs) [14] to directly operate on hypergraphs. We also collect more diverse VLSI hypergraph data to relieve the open-source data scarcity issue in the VLSI domain. The main contributions and results of this work are as follows.

- 1) To the best of our knowledge, we are the first to introduce HGNNs in hypergraph partitioning tasks, which can better capture the high-order relationships in hypergraphs. Based on this, a novel normalized cut loss computation algorithm designed for model training on GPU is proposed to directly optimize the partitioning quality based on HGNNs. It also significantly reduces GPU memory consumption compared to previous GNN-based methods.
- 2) We apply randomized techniques to accelerate eigenvalue decomposition during the node feature extraction stage, significantly improving the preprocessing efficiency.
- 3) We collect and open-source a comprehensive dataset of VLSI domain hypergraphs, facilitating future research in this area². Extensive experiments show our approach overall outperforms all types of existing methods on the ISPD98 and ISPD05 benchmarks, and training on our new dataset can further enhance its performance.

¹<https://github.com/THU-numbda/HGNN-Part>

²<https://github.com/THU-numbda/VLSI-Hypergraph-Benchmarks>

This work is supported by Beijing Natural Science Foundation (Z230002).

II. REVIEW OF HYPERGRAPH PARTITIONING PROBLEM

In this section, we first introduce the problem definition. Then we review some existing hypergraph partitioning methods, and briefly discuss their application in VLSI design.

A. Hypergraph Partitioning

A hypergraph $H = (V, E)$ consists of a vertex set V and a set of hyperedges E , where each hyperedge $e \in E$ is a non-empty subset of V . Hypergraphs generalize ordinary graphs by allowing edges (hyperedges) to connect arbitrary numbers of vertices. Given vertex weights w_v and hyperedge weights w_e , the balanced k -way hypergraph partitioning problem seeks to partition V into k disjoint blocks $\{V_1, \dots, V_k\}$ such that:

- Each block is balanced, i.e., $(\frac{1}{k} - \epsilon)W \leq \sum_{v \in V_i} w_v \leq (\frac{1}{k} + \epsilon)W$, $\forall 1 \leq i \leq k$, where $W = \sum_{v \in V} w_v$ and $\epsilon \in (0, \frac{1}{k})$ is a small imbalance tolerance.
- The objective is to minimize the total cutsize, typically defined as the sum of weights of hyperedges that span more than one block after the partitioning [15].

This problem is NP-hard [16], [17] and has numerous applications, including VLSI design, parallel scientific computing, and data mining [18]. In this paper, we assume $k = 2$ and uniform weights for all vertices and hyperedges for simplicity.

B. Partitioning Methods

The multilevel paradigm is still the most popular heuristic hypergraph partitioning method. It comprises of three main stages: 1) Coarsening: the hypergraph is recursively contracted by merging vertices or clusters, creating a hierarchy of smaller hypergraphs; 2) Initial Partitioning: a relatively good partition is computed on the coarsest hypergraph; 3) Refinement: the partition is projected back to finer levels, where it is incrementally refined usually via the Fiduccia-Mattheyses (FM) [19] algorithm or its variants [20]. Partitioners such as hMETIS [1], [2], PaToH [3], and KaHyPar [4] all follow this paradigm.

Spectral partitioning for hypergraphs seeks low-dimensional embeddings that reveal good partitions by solving eigenvalue problems [21], [22]. Traditional methods convert hypergraphs to graphs and apply Laplacian-based analysis [23]. Recent methods such as SpecPart [5] and K-SpecPart [6] formulate supervised spectral embeddings by solving a generalized eigenvalue problem: $\mathbf{L}\mathbf{x} = \lambda\mathbf{B}\mathbf{x}$, where \mathbf{L} is the Laplacian matrix and \mathbf{B} includes supervision from the initial solution and balance constraint. The computed eigenvectors are used to build a family of trees that distill the cut structure of the hypergraph and are subsequently employed to explore candidate solutions. These methods have demonstrated up to 50% cutsize reduction over traditional tools on VLSI benchmarks [5].

Deep learning has recently been employed for (hyper)graph partitioning. The GAP framework [7] formulates a continuous, differentiable relaxation of the normalized cut and balance loss, enabling fast, generalizable inference on unseen graphs. Multilevel evolutionary differentiable partitioners like MedPart [8] integrate GNNs into the multilevel framework, boosting solution quality by injecting learned global information.

GenPart [9] uses a variational graph generative model to learn a latent space that identifies high-quality vertex embeddings, and it improves the results through generative sampling and V-Cycle refinement. This enables diverse high-quality solutions and consistently outperforms hMETIS and even recent spectral methods in both cutsize and runtime on VLSI benchmarks.

C. Application in VLSI Design

Circuit netlists can be easily transformed into hypergraphs by mapping cells to vertices and nets to hyperedges, and smaller cutsize indicates smaller interconnection delays. Recent methods such as TritonPart [11] and EasyPart [12] introduce support for constraints relevant to modern VLSI design. TritonPart remains multilevel at its core but incorporates timing-aware partitioning by slack propagation. EasyPart includes new techniques for pursuing minimum hop during topology-driven partitioning and treating the interconnection constraints.

III. METHODOLOGY

This section presents the full methodology of our proposed framework, an HGNN-based approach for hypergraph partitioning. Fig. 1 provides an overview of the whole workflow, where the CPU and GPU usage is annotated for each stage. Our framework is based on GenPart [9], with three key improvements: (1) in the data preprocessing stage, we add new node features and introduce efficient randomized algorithms to accelerate the topology feature extraction process; (2) in the model architecture, we replace GNNs with HGNNs to better capture high-order relations in hypergraphs and reduce GPU memory consumption; (3) in the training procedure, we design a GPU-optimized algorithm for normalized cut loss computation, which is also explicitly tailored for hypergraphs.

A. Data Preprocessing and Feature Construction

For each vertex $v \in V$, GenPart constructs a feature vector $\mathbf{x}_v \in \mathbb{R}^5$, which is composed of four components: 1) Topology: the first two eigenvectors of the clique expansion graph \mathcal{G}_c [23], which are concatenated to form a $|V| \times 2$ feature matrix for all vertices; 2) Node Degree: the total number of vertices incident to vertex v ; 3) Pin Count: the total number of times v appears in all hyperedges; 4) Partition ID: An initial solution obtained via a single run of a multilevel partitioner such as hMETIS. Node masking is added to enhance generalization [24].

On this basis, we further enhance the feature set by introducing additional topological features that capture more comprehensive structural information of the hypergraph [25]. Specifically, we add computing the first two eigenvectors of the star expansion graph \mathcal{G}_s [23]. This is because \mathcal{G}_c only captures whether two vertices are connected by a hyperedge, but does not reflect the exact connectivity, i.e., how many hyperedges connect them. In contrast, \mathcal{G}_s introduces auxiliary nodes for each hyperedge, thus adding different connection paths between vertices. The difference is illustrated in the left part of Fig. 1. By combining features from both \mathcal{G}_c and \mathcal{G}_s , we can provide a more holistic representation of the hypergraph's structure.

Solving eigenvalue problems is often quite time-consuming, especially for large-scale hypergraphs with up to millions of

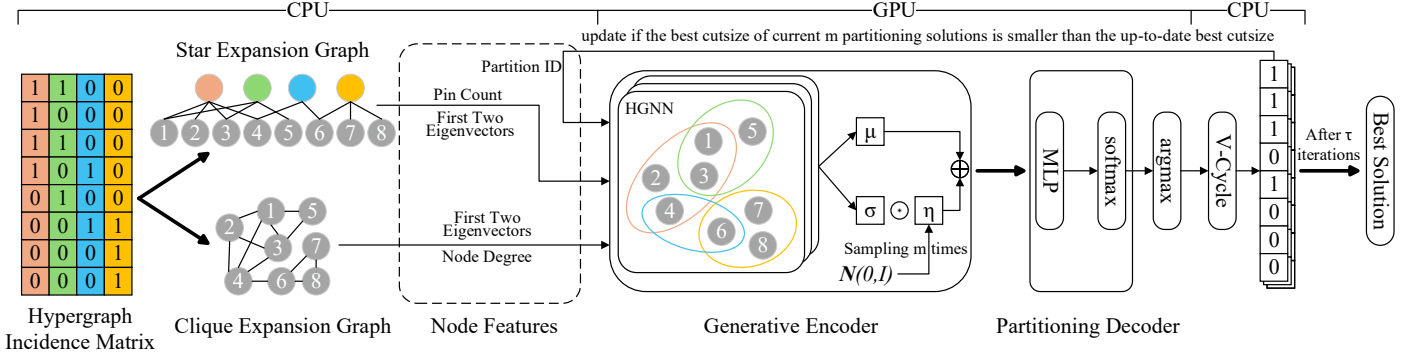


Fig. 1. Overview of the HGNN-Part workflow. Each stage is annotated with the computational resources used (CPU or GPU).

vertices and hyperedges. Now the introducing of \mathcal{G}_s doubles the computational cost. To address this, we exploit the special structure of the adjacency matrix and apply efficient approximate numerical methods. Let $\mathbf{H} \in \{0, 1\}^{|V| \times |E|}$ denotes the hypergraph incidence matrix, where $\mathbf{H}_{ij} = 1$ iff $v_i \in e_j$, then the adjacency matrix of the star expansion graph \mathcal{G}_s is:

$$\mathbf{A}_s = \begin{bmatrix} \mathbf{0} & \mathbf{H} \\ \mathbf{H}^\top & \mathbf{0} \end{bmatrix}.$$

Notice the eigenvectors of \mathbf{A}_s can be related to the singular vectors of \mathbf{H} . Specifically, if $\mathbf{A}_s \begin{bmatrix} \mathbf{u} \\ \mathbf{v} \end{bmatrix} = \lambda \begin{bmatrix} \mathbf{u} \\ \mathbf{v} \end{bmatrix}$, we can derive the following two equations:

$$\mathbf{H}\mathbf{v} = \lambda\mathbf{u}, \quad \mathbf{H}^\top\mathbf{u} = \lambda\mathbf{v}. \quad (1)$$

So, λ is the singular value of \mathbf{H} , and \mathbf{u}, \mathbf{v} are the corresponding singular vectors. Thus, computing the first two eigenvectors of \mathbf{A}_s can be transformed into running truncated SVD on a smaller graph \mathbf{H} . The first two left singular vectors of \mathbf{H} will compose two channels of the topological features.

Direct computation of the truncated SVD for large matrices is still computationally expensive compared to acquiring other features, so we utilize the randomized SVD in Algorithm 1. Its key idea is to introduce random projections to reduce the dimensionality of the input data while preserving its essential features. This significantly reduces computational cost with minimal accuracy loss as proven in [26].

Algorithm 1 Randomized Singular Value Decomposition

Input: Matrix $\mathbf{X} \in \mathbb{R}^{m \times n}$, target rank r , oversampling parameter s , power iteration number q

Output: $\mathbf{U}, \Sigma, \mathbf{V}$

- 1: $\Omega = \text{randn}(n, r + s)$
- 2: $\mathbf{Y} = \mathbf{X}\Omega$
- 3: $\mathbf{Q} = \text{orth}(\mathbf{Y})$
- 4: **for** $i = 1, \dots, q$ **do**
- 5: $\mathbf{T} = \text{orth}(\mathbf{X}^\top\mathbf{Q})$
- 6: $\mathbf{Q} = \text{orth}(\mathbf{X}\mathbf{T})$
- 7: **end for**
- 8: $\mathbf{B} = \mathbf{Q}^\top\mathbf{X}$
- 9: $\hat{\mathbf{U}}, \hat{\Sigma}, \hat{\mathbf{V}} = \text{svd}(\mathbf{B})$
- 10: $\mathbf{U} = \mathbf{Q}\hat{\mathbf{U}}[:, 1:r], \Sigma = \hat{\Sigma}[1:r, 1:r], \mathbf{V} = \hat{\mathbf{V}}[:, 1:r]$
- 11: **return** $\mathbf{U}, \Sigma, \mathbf{V}$

As for graph \mathcal{G}_c , its adjacency matrix does not exhibit the same sparsity patterns as \mathcal{G}_s , so we need to apply direct eigenvalue decomposition. However, we can still use the randomized method in Algorithm 2 to accelerate the computation, as proposed in [27]. In this paper, we set $s = 10$ and $q = 5$.

Algorithm 2 Randomized Eigenvalue Decomposition

Input: Matrix $\mathbf{X} \in \mathbb{R}^{n \times n}$, target rank r , oversampling parameter s , power iteration number q

Output: \mathbf{U}, Λ

- 1: $\Omega = \text{randn}(n, r + s)$
- 2: $\mathbf{Y} = \mathbf{X}\Omega$
- 3: $\mathbf{Q}, \sim, \sim = \text{eigSVD}(\mathbf{Y})$
- 4: **for** $i = 1, \dots, q$ **do**
- 5: $\mathbf{Q}, \sim, \sim = \text{eigSVD}(\mathbf{X}\mathbf{X}\mathbf{Q})$
- 6: **end for**
- 7: $\mathbf{S} = \mathbf{Q}^\top\mathbf{X}\mathbf{Q}$
- 8: $\hat{\mathbf{U}}, \Lambda = \text{eig}(\mathbf{S})$
- 9: $\mathbf{U} = \mathbf{Q}\hat{\mathbf{U}}$
- 10: **return** $\mathbf{U}[:, 1:r], \Lambda[1:r, 1:r]$

B. Model Architecture

Variational Graph Autoencoder (VGAE) [28] is a probabilistic generative framework that extends the principles of Variational Autoencoder (VAE) [29], [30] to graph-structured data. GenPart adopts VGAE as its core model, which can sample diverse vertex embeddings from a learned latent space to generate multiple partitioning solutions. However, VGAE is originally designed for normal graphs, so GenPart still applies clique expansion [23] in its implementation. This transformation not only leads to information loss, but also results in high memory consumption. As shown in Fig. 1, different from the previous stage, the encoder uses GPU rather than CPU, whose memory is more limited. Therefore, using clique expansion here would become a bottleneck in the whole workflow.

To overcome this, our core model consists of a variational hypergraph autoencoder (VHGAE), which extends VGAE to directly operate on hypergraphs. Its main component is the hypergraph neural network (HGNN) [14], which extends the graph neural network (GNN) [13] by supporting high-order (non-pairwise) relations. A typical HGNN layer is:

$$\mathbf{X}^{(l+1)} = \sigma \left(\mathbf{D}_v^{-1/2} \mathbf{H} \mathbf{W}_e \mathbf{D}_e^{-1} \mathbf{H}^\top \mathbf{D}_v^{-1/2} \mathbf{X}^{(l)} \Theta^{(l)} \right), \quad (2)$$

where $\mathbf{X}^{(l)}$ is the node feature matrix at layer l , \mathbf{H} is the incidence matrix, $\mathbf{D}_v, \mathbf{D}_e$ are diagonal degree matrices, \mathbf{W}_e is the hyperedge weight matrix, and σ is an activation function. This enables message passing from nodes to hyperedges and back, naturally capturing higher-order dependencies. It also largely reduces the memory usage, as \mathbf{H} is usually much sparser than the adjacency matrix of G_c . We will provide a detailed comparison in the last part of the experiment section.

Our encoder is composed of five HypergraphConv [31] layers, each with a hidden dimension of 256, except for the last layer with a dimension of 64 that yields the mean (μ) and standard deviation (σ) parameters of the latent space. The variational aspect is incorporated by sampling from the latent distribution using the reparameterization trick:

$$z = \mu + \sigma \odot \eta, \eta \sim \mathcal{N}(0, I).$$

Our decoder is the same as GenPart, mapping the latent representation z back to the input space via three linear layers (two with hidden dimension 64 and a final layer of dimension k followed by a softmax). During inference, m latent samples are drawn from the encoder, and the decoder produces m assignment probability matrices $\mathbf{Y} \in \mathbb{R}^{|V| \times k}$. Each vertex is assigned to the partition of maximum probability, yielding m initial solutions. As these solutions may not satisfy balance constraints, parallel V-Cycle refinement is applied. The best refined solution updates the partition ID feature if it improves the cutsize, and the procedure is iterated for τ rounds. In alignment with GenPart, we set $m = 12$ and $\tau = 11$.

C. Multi-Objective Training

Like GenPart, our training objective is a weighted sum of three loss terms, but we redesign the cut loss term to directly operate on hypergraphs, and revise the balance loss term.

1) *KL Divergence Loss*: The variational encoder is regularized by a KL-divergence term, which penalizes deviation from a standard normal distribution prior:

$$\mathcal{L}_{\text{KL}} = D_{\text{KL}}(\mathcal{N}(\mu, \sigma) \parallel \mathcal{N}(0, I)). \quad (3)$$

2) *Normalized Hyperedge Cut Loss*: Because now we formulate the model using hypergraphs, we have to redesign the cut loss term to directly compute the hyperedge cut using the incidence matrix \mathbf{H} . According to the definition of cutsize [15], the cut loss can be expressed as:

$$\mathcal{L}_{\text{Cut}} = |E| - \sum_{e \in E} \mathbb{E}[\exists 1 \leq i \leq k, e \subseteq V_i]. \quad (4)$$

It estimates the cutsize by calculating the expected number of hyperedges that are not cut, i.e., all their vertices belong to the same partition. However, larger hypergraphs with more hyperedges tend to have higher cut values, which can skew the loss, so we normalize the cut loss by the partition degrees:

$$\mathcal{L}_{\text{NCut}} = \mathcal{L}_{\text{Cut}} \times \sum_{i=1}^k \left(\sum_v \mathbf{D}_v \mathbf{Y}_{vi} \right)^{-1}, \quad (5)$$

where \mathbf{D} is the vertex degree vector. This formulation is scale-invariant and effective for large-scale hypergraphs. For

Algorithm 3 Normalized Cut Loss Computation

Input: Hypergraph incidence matrix $\mathbf{H} \in \{0, 1\}^{|V| \times |E|}$, partition assignment probability matrix $\mathbf{Y} \in \mathbb{R}^{|V| \times k}$, vertex degree vector $\mathbf{D} \in \mathbb{R}^{|V|}$

Output: Normalized cut loss $\mathcal{L}_{\text{NCut}}$

- 1: $(I_r, I_c) = \text{indices}(\mathbf{H})$
 - 2: $(I_c^{\text{sort}}, \pi) = \text{sort}(I_c)$
 - 3: $I_r^{\text{sort}} = I_r[\pi]$
 - 4: $\mathbf{Y}_{\text{exp}} = \mathbf{Y}[I_r^{\text{sort}}, :]$
 - 5: Initialize $\mathbf{P} = \mathbf{0} \in \mathbb{R}^{|E| \times k}$
 - 6: **for** $i = 1, \dots, k$ **do**
 - 7: $\mathbf{P}[:, i] = \text{scatter_mul}(\mathbf{Y}_{\text{exp}}[:, i], I_c^{\text{sort}})$
 - 8: **end for**
 - 9: $\mathcal{L}_{\text{Cut}} = |E| - \sum_{e=1}^{|E|} \sum_{i=1}^k \mathbf{P}[e, i]$, $\mathbf{d}_{\text{part}} = \sum_{v=1}^{|V|} \mathbf{D}[v] \cdot \mathbf{Y}[v, :]$
 - 10: $\mathcal{L}_{\text{NCut}} = \mathcal{L}_{\text{Cut}} \cdot \sum_{i=1}^k \frac{1}{\mathbf{d}_{\text{part}}[i]}$
 - 11: **return** $\mathcal{L}_{\text{NCut}}$
-

efficient GPU calculation, a scatter-multiplication strategy in `torch_scatter` package is used (see Algorithm 3), which avoids explicit net expansion and supports batched processing.

3) *Normalized Balance Loss*: To penalize unbalanced partitions, we use a normalized balance loss:

$$\mathcal{L}_{\text{Bal}} = \sum_{i=1}^k \left(\frac{\sum_v \mathbf{Y}_{vi} - \frac{|V|}{k}}{\frac{|V|}{k}} \right)^2. \quad (6)$$

The original balance loss in GenPart does not normalize the deviation by the expected block size, which can lead to instability when $|V|$ varies across hypergraphs. Our normalized version ensures consistent scaling and more stable training.

4) *Composite Loss*: The total loss is a weighted sum of the three loss components:

$$\mathcal{L} = \alpha \mathcal{L}_{\text{KL}} + \beta \mathcal{L}_{\text{NCut}} + \gamma \mathcal{L}_{\text{Bal}}, \quad (7)$$

with hyperparameters (α, β, γ) set by cross-validation or treated as learnable uncertainties. In the following experiments, we set $\alpha = 5 \times 10^{-4}, \beta = 0.5, \gamma = 100$.

IV. DATASET AND EXPERIMENTS

In this section, we present a new dataset and compare our method (HGNN-Part) with several state-of-the-art hypergraph partitioning methods on VLSI benchmarks. We implemented our method using Python, PyTorch and the PyTorch Geometric Framework (PyG). hMETIS is used as the partition ID feature provider and KaHyPar is used for V-Cycle refinement. For optimization, we use the Adam optimizer with a learning rate of 5×10^{-4} over 50 epochs. All the experiments were conducted on a Ubuntu 20.04 Linux server with an Intel Xeon Silver 4214 CPU and an NVIDIA GeForce RTX 4090 GPU.

A. A Comprehensive Dataset of VLSI Hypergraphs

GenPart [9] used ISPD98 benchmarks [32] as the training dataset and ISPD05 benchmarks [33] as the test dataset. To ensure a fair comparison, we basically follow the same setup. The characteristics of the benchmarks are summarized in Table I.

TABLE I
CHARACTERISTICS OF BENCHMARKS.

Source	Benchmark	V	E	Benchmark	V	E
ISPD98 [32]	IBM01	12,752	14,111	IBM10	69,429	75,196
	IBM02	19,601	19,584	IBM11	70,558	81,454
	IBM03	23,136	27,401	IBM12	71,076	77,240
	IBM04	27,507	31,970	IBM13	84,199	99,666
	IBM05	29,347	28,446	IBM14	147,605	152,772
	IBM06	32,498	34,826	IBM15	161,570	186,608
	IBM07	45,926	48,117	IBM16	183,484	190,048
	IBM08	51,309	50,513	IBM17	185,495	189,581
	IBM09	53,395	60,902	IBM18	210,613	201,920
ISPD05 [33]	adaptecl	211,447	221,142	bigblue1	278,164	284,479
	adaptecl2	255,023	266,009	bigblue2	557,866	577,235
	adaptecl3	451,650	466,758	bigblue3	1,096,812	1,123,170
	adaptecl4	496,045	515,951	bigblue4	2,177,353	2,229,886

In order to fully exploit the potential of deep learning technologies, we build a new dataset for hypergraph partitioning. We downloaded all available open-source VLSI design files from various EDA contests, including ISPD Contests [34]–[42], ICCAD Contest [43] and EDA Elite Challenge Contest [44], [45]. We also include the Titan23 benchmarks [46], which are famous for containing complex and real-world FPGA designs. All the files can be divided into three types of formats, and we use different open-source tools to parse them into hypergraphs:

- *Bookshelf Format*: This format is very close to the hypergraph representation, and we use a Bookshelf parser³ to map the nodes and nets to vertices and hyperedges.
- *LEF/DEF Format*: We first transformed these design files into Bookshelf format by using a LEF/DEF parser⁴ and then applied the previous Bookshelf parser.
- *Verilog Format*: We directly use the OpenROAD application⁵ to parse the Verilog files into hypergraphs.

We finally collected 164 hypergraphs from the VLSI field (including ISPD98 and ISPD05), and built a comprehensive dataset⁶, where the largest hypergraph has 3,066,539 vertices and 3,324,963 hyperedges. It will be used in the last experiment to show the result of using larger training data.

B. Experimental Results

We first compare our HGNN-Part with existing partitioning methods on the ISPD98 benchmarks (training dataset). We set the imbalance tolerance $\epsilon = 2\%$. The cutsizes results are summarized in Table II. The results of hMETIS, SpecPart, K-SpecPart, MedPart and GenPart are all directly cited from [9]. Our model uses both topological features from two types of expansion graphs and randomized eigenvalue/singular-value decomposition methods as the default setting (the last column). The table also includes the results of ablation studies to evaluate the performance of using different options on them.

From the results, we can see that our method achieves the best average improvement of 3.43%, which has an increase of 0.11% over GenPart, the previous state-of-the-art deep learning based method. Moreover, after comparing different feature combinations, we find out that using only topological features

³<https://github.com/PlebeianDev/Bookshelf-Format-Parser>

⁴https://github.com/jinwookjungs/lefdef_util

⁵<https://github.com/The-OpenROAD-Project/OpenROAD>

⁶<https://github.com/THU-numbda/VLSI-Hypergraph-Benchmarks>

TABLE II
CUTSIZE COMPARISON OF DIFFERENT PARTITIONING METHODS ON THE ISPD98 BENCHMARKS ($\epsilon = 2\%$, SETTING hMETIS AS THE BASELINE)

Benchmark	hMETIS	SpecPart	K-SpecPart	MedPart	GenPart	Ours			
						s	c	sc	sc+rand
IBM01	203	202	203	202	203	201	201	201	201
IBM02	354	336	333	339	327	327	327	326	326
IBM03	957	959	957	955	952	952	952	952	952
IBM04	595	593	580	583	580	579	579	580	579
IBM05	1733	1720	1716	1744	1706	1706	1706	1706	1706
IBM06	978	963	976	1000	964	963	963	963	964
IBM07	951	935	935	913	883	884	882	884	884
IBM08	1141	1146	1140	1146	1140	1140	1140	1140	1140
IBM09	629	620	620	623	620	620	620	620	620
IBM10	1333	1318	1257	1295	1254	1254	1254	1254	1254
IBM11	1071	1062	1051	1067	1051	1051	1051	1051	1051
IBM12	1982	1920	1937	1949	1920	1920	1920	1920	1920
IBM13	859	848	832	850	831	831	831	831	831
IBM14	1865	1859	1850	1876	1842	1842	1842	1842	1842
IBM15	2833	2741	2741	2855	2741	2741	2741	2741	2741
IBM16	2059	1915	1921	1972	1846	1881	2005	1842	1839
IBM17	2403	2354	2307	2336	2300	2294	2285	2286	2294
IBM18	1587	1535	1523	1587	1521	1573	1521	1521	1521
Avg Imp.	0%	1.83%	2.48%	1.12%	3.32%	3.12%	3.00%	3.43%	3.43%

s: only use topological features from the star expansion graph

c: only use topological features from the clique expansion graph

sc: use both topological features from star and clique expansion graphs

sc+rand: use both topological features and randomized svds/eigs methods (default)

from star expansion graph (s) or clique expansion graph (c) will both decrease the performance to different extents. This indicates that the two types of expansion graphs provide complementary global information for hypergraph partitioning. Additionally, employing randomized matrix decomposition methods (sc+rand) does not compromise the quality, demonstrating that our proposed acceleration techniques are effective and efficient. We validate that all the results meet the balance constraint.

As for runtime performance, we report the total partitioning time in seconds of hMETIS (run 10 times) and our method in Table III. We also break down our runtime into three parts: preprocess (node features extraction), reason (neural network inference), and refine (V-Cycle refinement). We compare the runtime of our method with and without randomized decomposition methods. From the results, we can see that the runtime of our method is comparable to hMETIS on most benchmarks. By analyzing the breakdown of the runtime, we find out that its main reason is the relatively slow speed of the V-Cycle stage provided by KaHyPar, which is quite sensitive to the initial solution. That also explains why our method does not necessarily

TABLE III
RUNTIME COMPARISON (IN UNIT OF SECOND) ($\epsilon = 2\%$)

Benchmark	hMETIS	Ours (randomized)				Ours (w/o randomized)			
		total	preprocess	reason	refine	total	preprocess	reason	refine
IBM01	8.9	12.1	1.7	0.9	9.5	14.9	2.6	0.9	11.4
IBM02	19.6	24.7	3.4	1.0	20.3	27.2	4.4	0.9	21.8
IBM03	21.4	20.5	3.0	1.0	16.5	23.3	4.6	0.9	17.7
IBM04	20.2	37.2	3.4	1.0	32.9	39.4	5.2	1.0	33.3
IBM05	30.2	37.3	5.7	1.0	30.6	44.0	7.4	1.0	35.7
IBM06	28.8	42.2	5.4	1.0	35.8	47.5	7.1	1.0	39.4
IBM07	43.9	113.3	5.2	1.0	107.2	123.6	10.1	1.0	112.5
IBM08	54.4	60.0	7.4	1.0	51.5	84.7	10.3	1.1	73.3
IBM09	37.1	78.3	5.7	1.1	71.4	91.6	10.4	1.1	80.1
IBM10	76.0	144.5	10.6	1.2	132.7	171.7	22.8	1.2	147.7
IBM11	63.6	313.9	7.9	1.1	304.8	322.8	13.5	1.2	308.1
IBM12	83.0	367.0	12.7	1.1	353.1	519.4	17.3	1.2	501.0
IBM13	73.0	175.3	8.9	1.2	165.2	215.0	17.9	1.2	195.9
IBM14	163.5	348.3	24.5	1.4	322.4	391.5	40.1	1.5	349.9
IBM15	194.1	760.9	23.8	1.6	735.5	1061.1	45.0	1.6	1014.5
IBM16	255.7	810.6	27.6	1.6	781.4	948.3	87.8	1.7	858.8
IBM17	299.4	474.8	45.0	1.9	427.9	672.1	80.8	1.8	589.6
IBM18	271.2	228.2	33.2	1.8	193.3	362.1	115.4	1.8	244.9

consume more time as the hypergraph size increases. Inference time is nearly negligible due to the fast computation speed of the GPU. Considering the preprocessing time, our randomized strategy exhibits a clear advantage, especially on large-scale hypergraphs, where it can save up to 80 seconds (about 25% of the total time) on the largest hypergraph.

To validate the generalization ability of our method, we also evaluate it on ISPD98 with a different ϵ value. To adapt to this change, we only have to change the imbalance parameter when acquiring the partition ID feature and when applying the V-Cycle refinement. We set $\epsilon = 10\%$ and the results are summarized in Table IV. Setting hMETIS as the baseline, our method still achieves the best average improvement of 4.85%, which has an increase of 0.06% over GenPart. All partitioning results are also validated to meet the new balance constraint.

Finally, we evaluate our method on the ISPD05 benchmarks (test dataset), with both $\epsilon = 2\%$ and $\epsilon = 10\%$. The results are summarized in Table V. To further validate the generalization capability of our approach, we train an enhanced variant of our method (Ours⁺) on our newly collected dataset (excluding the test dataset). Because this dataset contains much larger hypergraphs, we do the training on an NVIDIA A800 GPU but still run all the test processes in the default environment. From the results, we can see that our method still achieves the best average improvement of 5.02% ($\epsilon = 2\%$) and 1.97%

($\epsilon = 10\%$), which has an increase of 1.89% and 0.84% over GenPart, respectively. Moreover, by training on a larger dataset, our model further improves the cutsizes on average, demonstrating the effectiveness of our method in learning from more training data. All the partitioning results also meet the balance constraints of different ϵ values.

C. Discussion on GPU Memory Consumption

One advantage of our method is that we use hypergraphs to model the problem (HypergraphConv [31]) more efficiently, which saves much memory compared to the clique expansion graph (GraphConv [47]) used in GenPart. Here, we give an estimation of the GPU memory consumption for both methods.

Let $\text{nnz}(\mathbf{A}_c)$ and $\text{nnz}(\mathbf{H})$ denote the number of non-zero elements in the adjacency matrix of \mathcal{G}_c and hypergraph incidence matrix, respectively. Considering the largest-case scenario in terms of feature dimension, we use the hidden dimension (F_h) for memory estimation of input/output node features. Then, the GPU memory consumption of node features and necessary data for GraphConv and HypergraphConv can be estimated as $|V| \cdot F_h + 3 \cdot \text{nnz}(\mathbf{A}_c)$ and $|V| \cdot F_h + 3 \cdot \text{nnz}(\mathbf{H})$. Both GraphConv and HypergraphConv employ a mechanism called gather-scatter during convolution, and the GPU memory consumptions due to these intermediate results for GraphConv and HypergraphConv can be estimated as $\text{nnz}(\mathbf{A}_c) \cdot F_h$ and $\text{nnz}(\mathbf{H}) \cdot F_h$, respectively.

Thus, we can estimate the peak GPU memory consumption ratio of GraphConv to HypergraphConv approximately as:

$$\frac{|V| \cdot F_h + (3 + F_h) \cdot \text{nnz}(\mathbf{A}_c)}{|V| \cdot F_h + (3 + F_h) \cdot \text{nnz}(\mathbf{H})} \approx \frac{|V| + \text{nnz}(\mathbf{A}_c)}{|V| + \text{nnz}(\mathbf{H})}. \quad (8)$$

Empirical observations on the ISPD98 benchmarks indicate that this ratio ranges from 2.0 to 4.4. Therefore, under identical experiment settings, our method saves at least half of the GPU memory compared to GenPart. As a result, unlike GenPart which uses a 32GB V100 to run all the tests, we only need a 24GB RTX 4090. More obviously, GenPart cannot run some Titan23 instances due to memory limitations [9], while our method can handle them all, and reaches an improvement of 15.89% of cutsizes on average compared to hMETIS.

V. CONCLUSION

In this paper, a novel deep learning based approach for partitioning large-scale hypergraphs in VLSI design is presented. It utilizes hypergraph neural networks to capture high-order relationships in hypergraph, enabling more effective partitioning than traditional methods. Randomized matrix decomposition techniques are employed to enhance the efficiency of feature extraction without compromising the quality. Through extensive experiments on VLSI benchmarks, we demonstrated that our approach not only achieves superior partitioning quality but also significantly reduces GPU memory consumption. Training on a larger dataset further improves the model's performance, showcasing its ability to generalize well to unseen hypergraphs. In future work, we aim to investigate more advanced hypergraph neural network architectures and extend our method to arbitrary partition numbers k as well as hypergraphs with non-uniform vertex and hyperedge weights.

TABLE IV

CUTSIZE COMPARISON OF DIFFERENT PARTITIONING METHODS ON THE ISPD98 BENCHMARKS ($\epsilon = 10\%$, SETTING hMETIS AS THE BASELINE)

Benchmark	hMETIS	SpecPart	MedPart	GenPart	Ours
IBM01	190	171	166	166	166
IBM02	262	262	262	262	262
IBM03	960	952	954	950	950
IBM04	388	388	388	388	388
IBM05	1733	1688	1668	1645	1645
IBM06	760	733	760	733	732
IBM07	796	760	772	760	760
IBM08	1145	1140	1131	1120	1120
IBM09	535	519	520	519	519
IBM10	1284	1261	1257	1244	1248
IBM11	782	764	765	763	763
IBM12	1940	1842	1872	1841	1841
IBM13	721	693	696	655	655
IBM14	1665	1768	1605	1530	1516
IBM15	2262	2235	2166	2135	2123
IBM16	1708	1619	1645	1619	1619
IBM17	2300	1989	2024	1989	1989
IBM18	1550	1537	1550	1520	1520
Avg Imp.	0%	2.92%	3.27%	4.79%	4.85%

TABLE V

CUTSIZE COMPARISON OF DIFFERENT PARTITIONING METHODS ON THE ISPD05 BENCHMARKS (SETTING hMETIS AS THE BASELINE)

Benchmark	$\epsilon = 2\%$				$\epsilon = 10\%$			
	hMETIS	GenPart	Ours	Ours ⁺	hMETIS	GenPart	Ours	Ours ⁺
adaptec1	2790	2752	2576	2577	2234	2233	2154	2154
adaptec2	1153	1101	1100	1100	1113	1101	1100	1100
adaptec3	2804	2550	2467	2363	1947	1903	1921	1910
adaptec4	4140	4125	4028	4028	3445	3411	3349	3247
bigblue1	4217	4201	4201	4201	4133	4110	4110	4110
bigblue2	2179	2140	2121	2121	2199	2173	2121	2121
bigblue3	2859	2799	2708	2796	2652	2612	2617	2616
bigblue4	5655	5346	5381	5335	4902	4831	4830	4830
Avg Imp.	0%	3.13%	5.02%	5.20%	0%	1.13%	1.97%	2.42%

Ours⁺: train our model on the larger hypergraph dataset we collected

REFERENCES

- [1] G. Karypis, R. Aggarwal, V. Kumar, and S. Shekhar, "Multilevel hypergraph partitioning: Applications in VLSI domain," *IEEE Transactions on Very Large Scale Integration (VLSI) Systems*, vol. 7, no. 1, pp. 69–79, 1999.
- [2] G. Karypis and V. Kumar, "Multilevel k-way hypergraph partitioning," in *Proceedings of the 36th annual ACM/IEEE design automation conference*, 1999, pp. 343–348.
- [3] Ü. V. Çatalyürek and C. Aykanat, "PaToH (partitioning tool for hypergraphs)," 2011.
- [4] S. Schlag, T. Heuer, L. Gottesbüren, Y. Akhremtsev, C. Schulz, and P. Sanders, "High-quality hypergraph partitioning," *ACM Journal of Experimental Algorithmics*, vol. 27, pp. 1–39, 2023.
- [5] I. Bustany, A. B. Kahng, I. Koutis, B. Pramanik, and Z. Wang, "SpecPart: A supervised spectral framework for hypergraph partitioning solution improvement," in *Proceedings of the 41st IEEE/ACM International Conference on Computer-Aided Design*, 2022, pp. 1–9.
- [6] I. Bustany, A. B. Kahng, I. Koutis, B. Pramanik, and Z. Wang, "K-SpecPart: Supervised embedding algorithms and cut overlay for improved hypergraph partitioning," *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, 2023.
- [7] A. Nazi, W. Hang, A. Goldie, S. Ravi, and A. Mirhoseini, "GAP: Generalizable approximate graph partitioning framework," *arXiv preprint arXiv:1903.00614*, 2019.
- [8] R. Liang, A. Agnesina, and H. Ren, "MedPart: A multi-level evolutionary differentiable hypergraph partitioner," in *Proceedings of the 2024 International Symposium on Physical Design*, 2024, pp. 3–11.
- [9] M. Chen and T.-C. Wang, "A hypergraph partitioner utilizing a novel graph generative model," in *Proceedings of the 43rd IEEE/ACM International Conference on Computer-Aided Design*, 2024, pp. 1–9.
- [10] M. H. Khan, B. Onal, E. Dogan, and M. R. Guthaus, "VLSI hypergraph partitioning with deep learning," *arXiv preprint arXiv:2409.01387*, 2024.
- [11] I. Bustany, G. Gasparyan, A. B. Kahng, I. Koutis, B. Pramanik, and Z. Wang, "An open-source constraints-driven general partitioning multi-tool for VLSI physical design," in *2023 IEEE/ACM International Conference on Computer Aided Design (ICCAD)*. IEEE, 2023, pp. 1–9.
- [12] S. Tong, H. Li, J. Xu, C. Pei, W. Yu, S. Liu, and J. Shen, "EasyPart: An effective and comprehensive hypergraph partitioner for FPGA-based emulation," in *Proceedings of the 43rd IEEE/ACM International Conference on Computer-Aided Design*, 2024, pp. 1–9.
- [13] F. Scarselli, M. Gori, A. C. Tsoi, M. Hagenbuchner, and G. Monfardini, "The graph neural network model," *IEEE transactions on neural networks*, vol. 20, no. 1, pp. 61–80, 2008.
- [14] Y. Feng, H. You, Z. Zhang, R. Ji, and Y. Gao, "Hypergraph neural networks," in *Proceedings of the AAAI conference on artificial intelligence*, vol. 33, no. 01, 2019, pp. 3558–3565.
- [15] C. J. Alpert and A. B. Kahng, "Recent directions in netlist partitioning: A survey," *Integration*, vol. 19, no. 1-2, pp. 1–81, 1995.
- [16] D. S. Johnson and M. R. Garey, *Computers and intractability: A guide to the theory of NP-completeness*. WH Freeman, 1979.
- [17] T. Lengauer, *Combinatorial algorithms for integrated circuit layout*. Springer Science & Business Media, 2012.
- [18] S. Tong, C. Pei, and W. Yu, "BlasPart: A deterministic parallel partitioner for balanced large-scale hypergraph partitioning," in *2025 62nd ACM/IEEE Design Automation Conference*. IEEE, 2025, pp. 1–6.
- [19] C. M. Fiduccia and R. M. Mattheyses, "A linear-time heuristic for improving network partitions," in *Papers on Twenty-five years of electronic design automation*, 1988, pp. 241–247.
- [20] J. Cong and S. K. Lim, "Multiway partitioning with pairwise movement," in *Proceedings of the 1998 IEEE/ACM international conference on Computer-aided design*, 1998, pp. 512–516.
- [21] J. R. Lee, S. O. Gharan, and L. Trevisan, "Multiway spectral partitioning and higher-order cheeger inequalities," *Journal of the ACM (JACM)*, vol. 61, no. 6, pp. 1–30, 2014.
- [22] T.-H. H. Chan, A. Louis, Z. G. Tang, and C. Zhang, "Spectral properties of hypergraph laplacian and approximation algorithms," *Journal of the ACM (JACM)*, vol. 65, no. 3, pp. 1–48, 2018.
- [23] L. Hagen and A. B. Kahng, "New spectral methods for ratio cut partitioning and clustering," *IEEE transactions on computer-aided design of integrated circuits and systems*, vol. 11, no. 9, pp. 1074–1085, 1992.
- [24] J. Devlin, M.-W. Chang, K. Lee, and K. Toutanova, "BERT: Pre-training of deep bidirectional transformers for language understanding," in *Proceedings of the 2019 conference of the North American chapter of the association for computational linguistics: human language technologies, volume 1 (long and short papers)*, 2019, pp. 4171–4186.
- [25] Z. Zhang, P. Cui, J. Pei, X. Wang, and W. Zhu, "Eigen-GNN: A graph structure preserving plug-in for GNNs," *IEEE Transactions on Knowledge and Data Engineering*, vol. 35, no. 3, pp. 2544–2555, 2021.
- [26] N. Halko, P.-G. Martinsson, and J. A. Tropp, "Finding structure with randomness: Probabilistic algorithms for constructing approximate matrix decompositions," *SIAM review*, vol. 53, no. 2, pp. 217–288, 2011.
- [27] Y. Xie, Y. Dong, J. Qiu, W. Yu, X. Feng, and J. Tang, "SketchNE: Embedding billion-scale networks accurately in one hour," *IEEE Transactions on Knowledge and Data Engineering*, vol. 35, no. 10, pp. 10 666–10 680, 2023.
- [28] T. N. Kipf and M. Welling, "Variational graph auto-encoders," *arXiv preprint arXiv:1611.07308*, 2016.
- [29] D. P. Kingma and M. Welling, "Auto-encoding variational bayes," *CoRR*, vol. abs/1312.6114, 2013.
- [30] D. J. Rezende, S. Mohamed, and D. Wierstra, "Stochastic backpropagation and approximate inference in deep generative models," in *International conference on machine learning*. PMLR, 2014, pp. 1278–1286.
- [31] S. Bai, F. Zhang, and P. H. Torr, "Hypergraph convolution and hypergraph attention," *Pattern Recognition*, vol. 110, p. 107637, 2021.
- [32] C. J. Alpert, "The ISPD98 circuit benchmark suite," in *Proceedings of the 1998 international symposium on Physical design*, 1998, pp. 80–85.
- [33] G.-J. Nam, C. J. Alpert, P. Villarrubia, B. Winter, and M. Yildiz, "The ISPD2005 placement contest and benchmark suite," in *Proceedings of the 2005 international symposium on Physical design*, 2005, pp. 216–220.
- [34] G.-J. Nam, C. Aplert, and P. G. Villarrubia, "The ISPD 2006 placement contest and benchmark suite," in *Slides presented at ISPD'06*, 2006.
- [35] V. Yutsis, I. S. Bustany, D. Chinnery, J. R. Shinnerl, and W.-H. Liu, "ISPD 2014 benchmarks with sub-45nm technology rules for detailed-routing-driven placement," in *Proceedings of the 2014 on International symposium on physical design*, 2014, pp. 161–168.
- [36] I. S. Bustany, D. Chinnery, J. R. Shinnerl, and V. Yutsis, "ISPD 2015 benchmarks with fence regions and routing blockages for detailed-routing-driven placement," in *Proceedings of the 2015 Symposium on International Symposium on Physical Design*, 2015, pp. 157–164.
- [37] S. Yang, A. Gayasen, C. Mulpuri, S. Reddy, and R. Aggarwal, "Routability-driven FPGA placement contest," in *Proceedings of the 2016 on International Symposium on Physical Design*, 2016, pp. 139–143.
- [38] C.-W. Pui, G. Chen, Y. Ma, E. F. Young, and B. Yu, "Clock-aware ultrascale FPGA placement with machine learning routability prediction," in *2017 IEEE/ACM International Conference on Computer-Aided Design (ICCAD)*. IEEE, 2017, pp. 929–936.
- [39] S. Mantik, G. Posser, W.-K. Chow, Y. Ding, and W.-H. Liu, "ISPD 2018 initial detailed routing contest and benchmarks," in *Proceedings of the 2018 International Symposium on Physical Design*, 2018, pp. 140–143.
- [40] W.-H. Liu, S. Mantik, W.-K. Chow, Y. Ding, A. Farshidi, and G. Posser, "ISPD 2019 initial detailed routing contest and benchmark with advanced routing rules," in *Proceedings of the 2019 international symposium on physical design*, 2019, pp. 147–151.
- [41] J. Knechtel, J. Gopinath, M. Ashraf, J. Bhandari, O. Sinanoglu, and R. Karri, "Benchmarking security closure of physical layouts: ISPD 2022 contest," in *Proceedings of the 2022 International Symposium on Physical Design*, 2022, pp. 221–228.
- [42] M. Eslami, J. Knechtel, O. Sinanoglu, R. Karri, and S. Pagliarini, "Benchmarking advanced security closure of physical layouts: ISPD 2023 contest," in *Proceedings of the 2023 International Symposium on Physical Design*, 2023, pp. 256–264.
- [43] B.-Y. Wu, R. Liang, G. Pradipta, A. Agnesina, H. Ren, and V. A. Chhabria, "2024 ICCAD CAD contest problem c: Scalable logic gate sizing using ML techniques and GPU acceleration," in *Proceedings of the 43rd IEEE/ACM International Conference on Computer-Aided Design*, 2024, pp. 1–5.
- [44] S2C, "2023 integrated circuit EDA elite challenge contest," 2023. [Online]. Available: <http://edachallenge.cn>
- [45] S2C, "2024 China postgraduate ic innovation competition · EDA elite challenge contest," 2024. [Online]. Available: <http://edachallenge.cn>
- [46] K. E. Murray, S. Whitty, S. Liu, J. Luu, and V. Betz, "Titan: Enabling large and complex benchmarks in academic CAD," in *2013 23rd International Conference on Field Programmable Logic and Applications*. IEEE, 2013, pp. 1–8.
- [47] C. Morris, M. Ritzert, M. Fey, W. L. Hamilton, J. E. Lenssen, G. Rattan, and M. Grohe, "Weisfeiler and leman go neural: Higher-order graph neural networks," in *Proceedings of the AAAI conference on artificial intelligence*, vol. 33, no. 01, 2019, pp. 4602–4609.