

FGNN2: A Powerful Pre-training Framework for Learning the Logic Functionality of Circuits

Ziyi Wang, Chen Bai, Zhuolun He, Guangliang Zhang, Qiang Xu, Tsung-Yi Ho, Yu Huang, Bei Yu

Abstract—Learning feasible representation from raw gate-level circuits is essential for incorporating machine learning techniques in logic synthesis, physical design, or verification. Existing structure-based learning methods tend to concentrate mainly on graph topology, often neglecting logic functionality. This oversight frequently results in a failure to capture the underlying semantics, thereby limiting their overall applicability. To address the concern, we propose a novel circuit representation learning framework, FGNN2, that utilizes a contrastive scheme to effectively extract generic functionality knowledge. We construct a comprehensive pre-training dataset through a customized circuit augmentation scheme. We have also developed a novel contrastive loss function to capture the relative functional distance between different circuits, and to generate representations that are invariant to the input order. In addition, we employed a customized graph neural network (GNN) architecture to better align with the above framework. Comprehensive experiments on multiple complex real-world designs demonstrate that our proposed solution significantly outperforms state-of-the-art circuit representation learning flows.

I. INTRODUCTION

With the rapid advancement of machine learning (ML) techniques, there has been a marked increase in the integration of ML into electronic design automation (EDA) [1]–[5]. Most of these studies employ a representation learning approach that initially learns low-dimensional representations from high-dimensional raw data, and then performs classification or regression based on these learned representations. The overall effectiveness of this framework is contingent upon the quality of the learned representations. Our study is focused on representation learning for gate-level electronic circuits, a fundamental object in EDA. This is non-trivial as a circuit contains various components, which might vary largely in structures and connectivity.

In recent years, the fast-growing deep neural network techniques have shown great power in circuit representation learning [6]–[10]. For instance, a compact representation termed level-dependent decaying sum (LDDS) existence vector (EV) is introduced in [6] to embed a circuit node with its neighbors. Ma et al. [7] propose an iterative process to insert observation points into gate-level circuits based on the node representations learned by a graph neural network (GNN). The

The preliminary version has been presented at the ACM/IEEE Design Automation Conference (DAC) in 2022. This work is partially supported by HiSilicon and The Research Grants Council of Hong Kong SAR CUHK14209420, CUHK14208021. (Corresponding authors: Bei Yu)

Z. Wang, C. Bai, Z. He, Q. Xu, T.Y. Ho and B. Yu are with the Department of Computer Science and Engineering, The Chinese University of Hong Kong, Hong Kong SAR.

G. Zhang and Y. Huang are with Huawei HiSilicon.

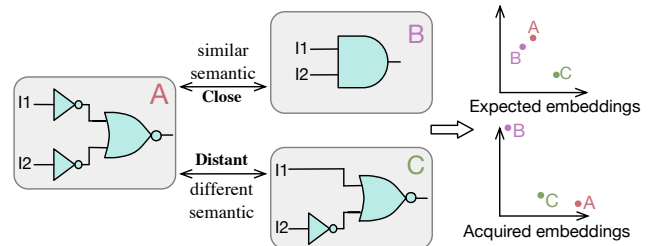


Fig. 1 Illustration of the drawback of existing structure-based circuit representation learning methods.

work of [8] develops a graph learning-based solution to extract desired logic components from a flattened circuit, where a novel graph neural network customized for directed acyclic graph (DAG) is proposed to generate gate representations.

Though the above circuit representation learning methods have achieved state-of-the-art performance in many circuit-level tasks, we argue that they are still not sufficient. One primary concern is the limited emphasis on Boolean functionality in existing methods, despite its crucial role in understanding electronic circuits. Typically, GNN-based approaches incorporate functional information like gate type into the node attributes to implicitly learn circuit functionality. However, this implicit approach may result in the loss of essential functional information, thereby lacking the ability to extract robust circuit representations. Furthermore, these methods are usually tailored for a specific downstream task. Consequently, they usually suffer from poor generalization ability. Fig. 1 gives an example to illustrate the limitations of existing structure-based circuit representation learning methods. Circuits A and B, which perform the same function and share similar semantics, should ideally be close in the representation space. However, due to their topological differences, structural methods tend to distance their representations. Conversely, Circuits A and C, which perform different functions and should be far apart in the representation space, are drawn together by existing methods due to their structural similarities. This example shows that the structure-based learning methods often fail to capture the real semantics of circuits accurately.

Fortunately, we have observed a growing academic interest in learning circuit representations with enhanced generalization capabilities [11], [12]. These studies aim to learn the logic functionality of circuits, which is consistent across various designs, rather than focusing on the unstable structural information. They typically follow a self-supervised pre-training paradigm with task-agnostic supervision to achieve

this goal. One prominent solution, known as **FGNN** [11], focuses on distinguishing between functionally equivalent and inequivalent circuits through contrastive learning. This approach introduces a unique circuit augmentation scheme that introduces structural perturbations while preserving the boolean functionality of the circuits. However, a significant limitation of this approach is the direct application of the standard contrastive **loss function**, which fails to accurately capture the relative distance between different negative samples and the same target sample. Moreover, the limited structural variance introduced during the data augmentation process hinders its robust generalization ability. Another representative work is Deepgate2 [12], which utilizes pairwise truth table differences between sampled logic gates as training supervision. Nevertheless, this method employs a simplistic loss function that only considers a pair of negative samples from a fixed set of pre-sampled circuit pairs at each time. Consequently, it struggles to accurately differentiate circuits with different functionalities, especially those with similar structures like adders and subtractors. Furthermore, Deepgate2 can only capture the relationship between circuits with the same input width, which limits its comprehension of logic functionality. It is also important to note that Deepgate2 is sensitive to the order of the circuit's input signals, which could potentially limit its practical application.

To address the above concerns, we introduce FGNN2, an innovative pre-training framework for circuit functionality learning that significantly enhances the original FGNN. One of the key advancements in FGNN2 is an improved contrastive loss function specifically tailored for circuit functionality learning. This loss function is capable of effectively capturing the relative functional distance between different circuits and generating circuit embeddings that are invariant to input orders, which are crucial in practical applications. Alongside the refined loss function, we have also proposed a more robust circuit augmentation scheme. This scheme generates a substantially more comprehensive pre-training dataset, characterized by increased structural diversity. Moreover, we consider the comparison of circuits with different input widths when constructing the dataset. By leveraging this augmented scheme, we have acquired a pre-training dataset with superior diversity compared to previous methods such as FGNN and DeepGate2. Consequently, FGNN2 exhibits a deeper understanding of logic functionality and demonstrates improved generalization capabilities.

Our major contributions are summarized as follows:

- We present a novel contrastive learning-based pre-training framework customized for gate-level circuits, which allows for the extraction of universal logic functionality knowledge.
- We construct a large dataset for circuit functionality learning through a novel circuit augmentation scheme.
- We introduce a customized contrastive loss function capable of capturing the relative functional distance between different circuits. Furthermore, the circuit representations produced are invariant to the input order.
- We conduct comprehensive experiments on several complex real-world designs, which confirms the effectiveness

of our proposed framework compared with state-of-the-art circuit representation learning arts.

- Our circuit pre-training framework FGNN2 together with our synthetic dataset has been **open-sourced** [13].

The rest of the paper is organized as follows: Section II introduces some preliminaries. Section III gives the problem formulation and the overview of our proposed circuit representation learning framework. Section IV introduces the pre-training dataset generation through a customized circuit augmentation scheme. Section V introduces our novel circuit contrastive learning scheme. Section VI presents experimental evaluations of our proposed framework. Section VII concludes the paper.

II. PRELIMINARIES

A. Graph Neural Network

In recent years, Graph Neural Networks (GNNs) have emerged as a promising approach for analyzing graph-structured data [14]–[16]. They employ an iterative neighborhood aggregation scheme to capture the structural information within the neighborhoods of nodes. Let $G = \langle \mathcal{V}, \mathcal{E} \rangle$ denotes a graph, where $\mathcal{V} = \{v_1, v_2, \dots, v_n\}$ is the vertex set, and $\mathcal{E} \subseteq \mathcal{V} \times \mathcal{V}$ is the edge set. Considering a K-layer GNN, the propagation of the k -th layer is represented as

$$\begin{aligned} \mathbf{m}_v^{(k)} &= \text{AGGREGATE}(\{\mathbf{h}_u^{(k-1)} : u \in \mathcal{N}(v)\}), \\ \mathbf{h}_v^{(k)} &= \text{COMBINE}(\mathbf{m}_v^{(k)}, \mathbf{h}_v^{(k-1)}), \end{aligned} \quad (1)$$

where $\mathbf{h}_v^{(k)}$ is the representation vector of vertex v at the k -th layer. $\mathcal{N}(v)$ denotes the neighboring nodes of v , and AGGREGATE is a function used to collect messages from a node's neighborhood. COMBINE is leveraged to combine the node's previous representation with its neighborhood message.

Several GNN variants [17]–[22] have been proposed, achieving state-of-the-art performance in related graph learning tasks. Particularly, there is a growing interest in directed acyclic graphs (DAGs) [23], [24], which are widely used to model real-world data, including gate-level circuits. To generate better global-level embeddings for DAGs, the work of [23] proposes an impressive GNN architecture that leverages the partial order induced by DAGs. Additionally, the work of [24] introduces an asynchronous message passing scheme to encode computation graphs and develop a variational autoencoder for DAGs, known as D-VAE.

B. Graph Contrastive Learning

Contrastive learning (CL) is employed as a new paradigm for model pre-training. The core concept of contrastive learning involves capturing both relevant and irrelevant statistical dependencies by maximizing the separation between positive and negative samples in the n -dimensional embedding space \mathbb{R}^n . The objective of CL is to learn an encoder $f : x \rightarrow e, e \in \mathbb{R}^n$ that for any sample x :

$$\text{score}(f(x), f(x^+)) \gg \text{score}(f(x), f(x^-)). \quad (2)$$

Here x^+ represents positive samples that are similar or identical to x , typically obtained through data augmentation.

On the other hand, x^- refers to negative samples that differ from x . The function $score(e_1, e_2)$ measures the similarity (distance) between two embeddings e_1 and e_2 .

Due to its outstanding performance, contrastive learning has achieved remarkable success in the field of computer vision [25]–[28]. Theoretical analyses have shed light on the underlying reasons for this success [29], demonstrating that the objectives employed in contrastive methods can be viewed as maximizing a lower bound of the mutual information between the input data and the output representations.

In recent years, researchers have been actively exploring the extension of contrastive methods to handle graph data. For instance, DGI [30] incorporates high-order global contextual features into node representations by maximizing the mutual information between global and local embeddings. The work of [31] generates multiple views of a graph by introducing various perturbations, such as edge dropping, node dropping, and feature masking. GCA [32] is proposed to further explore graph data augmentation methods.

However, most existing graph contrastive learning methods only take the structural information into account during the data augmentation procedures, leading to inadequate or even wrong understanding of the target graphs. We argue that structural information is not always consistent with the semantics in many scenarios. For example, two netlists with different structures may serve the same function, e.g., a ripple-carry adder and a carry look-ahead adder. Additionally, even a minor perturbation can completely alter the semantics of a graph, e.g., removing a single wire connection can change the function of a circuit. Consequently, existing methods fail to ensure the consistency between the augmented view and the original input, as they introduce random structural perturbations (e.g., random edge/node dropping) that may alter the underlying semantics.

III. PROBLEM FORMULATION & FRAMEWORK OVERVIEW

We first introduce the gate-level circuit and then give the problem formulation. The gate-level circuit consists of a list of gate-level components (e.g., AND gates) and interconnects (wires) between them. Gate-level circuits are generated by converting a description of circuit behavior at the register transfer level (RTL) into design implementation in logic gates. A gate-level netlist can be formulated as a directed acyclic graph (DAG) with nodes denoting circuit components and edges representing wires. Based on the definition of the gate-level circuit, our problem can be formulated as follows:

Problem 1 (Circuit Functionality Learning). *Design a novel learning methodology that automatically discovers gate/circuit representations capturing their boolean functional semantics. We hope the representation facilitates multiple downstream gate-level circuit tasks, covering: (1) local scenario, e.g., identifying desired components (viz., sub-circuits) located in the circuit, and (2) global scenario, e.g., classifying the circuits into one of the categories according to its functionality.*

Before diving into the specifics of the algorithms involved, we provide a concise overview of our proposed approach for

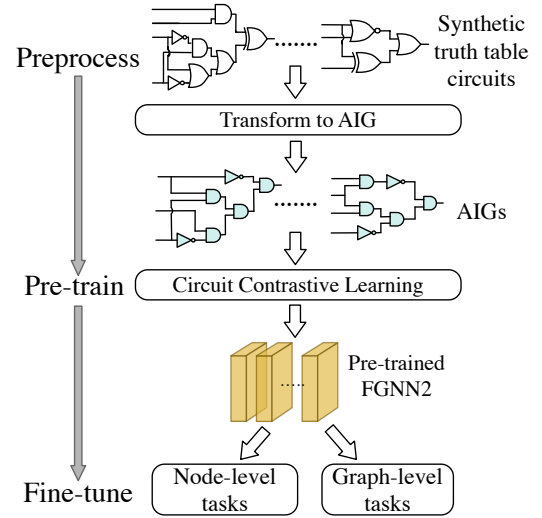


Fig. 2 Our proposed circuit representation learning flow

circuit functionality learning, as depicted in Fig. 2. In general, our method follows the **pre-training** fashion that involves training the model on a large dataset to learn general features before fine-tuning it on a specific task or dataset. The entire process can be summarized into three steps:

1. Preprocessing: To begin with, we construct an extensive synthetic circuit dataset for pre-training (refer to Section IV). Each circuit within this dataset represents a truth table and is transformed into the And-Inverter Graph (AIG) form.

2. Pre-training: Subsequently, we employ a novel circuit contrastive learning scheme on the generated dataset to extract fundamental knowledge about logic functionality. This is achieved by utilizing a tailored contrastive loss function (refer to Section V-A), along with a specially-designed functional graph neural network (FGNN) (refer to Section V-B) as the encoder.

3. Fine-tuning: Lastly, the pre-trained GNN is equipped with a Multilayer Perceptron (MLP)-based classifier and fine-tuned to adapt downstream tasks.

IV. CIRCUIT AUGMENTATION AND DATASET GENERATION

Previous studies have demonstrated that the efficacy of contrastive learning hinges on the assumption that important information is shared across different views of the same sample [33], [34]. This implies that the semantics of augmented views should remain consistent with that of the original sample.

While generating augmented views without causing semantic changes is relatively straightforward for images (e.g., translation, scale) [26], it is not as explicit for graph data. The primary challenge lies in the fact that the semantics of graphs are often not apparent, and even minor perturbations can completely alter their meaning. For instance, the addition of a single connection between two gates can drastically change the functionality of a circuit, leading to bugs and errors. In order to address this concern and extend the contrastive learning scheme to accommodate circuits, it is crucial to

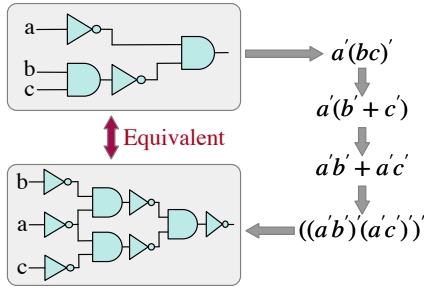


Fig. 3 Illustration of Boolean Equivalence

explore a tailored data augmentation approach that introduces perturbations without compromising the original semantics.

Taking inspiration from the domain knowledge of Electronic Design Automation (EDA), we design a novel circuit augmentation scheme that alters the topology structure of a given circuit while preserving its underlying semantics. To begin with, we define the semantics of an electronic circuit as its logic functionality, which determines the output behavior of the circuit. Formally, the logic functionality of a circuit can be defined as the Boolean expressions of its output signals, as illustrated in Fig. 3. We consider two circuits to possess similar semantics if their output Boolean expressions exhibit similarity. Then the problem becomes how to alter the circuit structure without changing its logic functionality.

Fortunately, there is a powerful tool within the realm of Boolean Algebra that precisely meets our requirements: the equivalent Boolean transformation. This technique allows us to convert one Boolean expression into another, despite having entirely different forms, while preserving the same logic functionality. As a result, circuits implementing these transformed Boolean expressions possess the same semantics but exhibit distinct structures, as depicted in Fig. 3. Consequently, they can be regarded as perfect augmented views of one another.

Building upon this observation, we have developed a novel circuit augmentation scheme to maximize the mutual information between circuits with similar functionalities. Our scheme initiates by transforming the target circuits into a unified format known as And-Inverter Graph (AIG) using the open-source logic synthesis tool ABC [35]. We perform such transformation based on three considerations: (1) It enhances the generalization capability of our model since AIG is a unified format; (2) It simplifies the learning process by limiting our focus to two types of gates, as opposed to a myriad of types. (3) It facilitates the generation of a more comprehensive pre-training dataset, given that we can leverage the well-established rewriting algorithm in ABC. From the results listed in TABLE IV and TABLE VII, it can be seen that the transformation from gate-level netlist (FGNN) to AIG (FGNN2) actually brings performance benefits.

After the above transformation, we generate augmented views for each circuit through logic rewriting, a widely employed logic synthesis technique that replaces sub-circuits with more efficient structures while preserving the original logic functionality. Instead of directly utilizing the rewriting operation in ABC, we have implemented a modified version

that incorporates a customized gain function, taking into account structural diversity. Our customized rewriting algorithm is illustrated in Algorithm 1.

Following the ABC framework, our rewriting algorithm iterates through cut enumeration (line 3), evaluation (lines 4-10), and replacement (lines 13-17) for each node sequentially in the topological order. For each identified cut during cut enumeration, we search for candidate subgraphs V that are logically equivalent to its logic cone from a pre-computed library. Subsequently, these subgraphs are evaluated using the customized gain function depicted in Equation (3), where α is a hyperparameter, $|S|$ is the number of nodes in subgraph S , and $O(S)$ denotes the occurrence count of S throughout the whole rewriting process. Generally, we prioritize subgraphs that exhibit greater variance (larger $|S|$) but have been utilized less frequently in previous attempts. During the evaluation process, the cut and the corresponding subgraph that yields the highest gain are retained and denoted as BestC and BestS, respectively. Once all the cuts of the node have been visited, we replace the logic cone of BestC with the subgraph BestS and then increment the usage count of BestS. The entire rewriting process terminates when either all nodes have been visited or the replacement ratio $R = N_r/|AIG|$, which is defined as the proportion of replaced nodes, exceeds a pre-defined threshold T .

$$Gain(S) = |S| - \alpha O(S) \quad (3)$$

It is important to note that our rewriting algorithm is highly efficient as it employs the same greedy strategy used in ABC, i.e., traversing nodes in a topological order and attempting to replace each node's rooted AIG subgraph with pre-computed subgraphs. Here we provide a concise analysis of the time complexity of our algorithm. Let \mathcal{E} denote the number of edges, and \mathcal{N} denote the number of nodes. The runtime bottleneck of the algorithm lies in enumerating the 4-feasible cuts for all nodes using the procedure in [36], which can be finished in $\mathcal{O}(\mathcal{E})$ time. In the worst case, our algorithm would need to deal with all the 4-feasible cuts, totaling $\mathcal{O}(\mathcal{N})$ in number. For each cut, our algorithm involves enumerating the candidate pre-computed subgraphs, with a time complexity of $\mathcal{O}(1)$ per cut. Consequently, the overall time complexity of our algorithm is $\mathcal{O}(\mathcal{E} + \mathcal{N})$. Moreover, our pre-trained circuits are typically sparsely connected directed acyclic graphs, implying that \mathcal{E} is close to \mathcal{N} . As a result, the time complexity of our algorithm approximates $\mathcal{O}(\mathcal{N})$.

Based on our circuit argumentation scheme, a large synthetic dataset is constructed for pre-training purposes. The construction process begins by generating a substantial number of one-output truth tables, with the number of inputs ranging from 4 to 7. Each truth table in the dataset represents a specific Boolean function. The representative node of each truth table is the sole output node, and its embedding is utilized to represent the functionality of the entire truth table circuit. Subsequently, we run the logic synthesis tool ABC with our customized rewriting technique to transform the circuits into And-Inverter Graphs (AIGs) and generate the augmented views for each circuit.

TABLE I Dataset statistic showing the number of training samples, where width represents the number of inputs.

Width	Replacement Ratio				total
	10%	20%	30%	50%	
4	5666	5548	4302	3924	19440
5	200000	200000	200000	200000	800000
6	200000	200000	200000	200000	800000
7	200000	200000	200000	200000	800000
total	605666	605548	604302	603924	2401440

In order to enhance the pre-trained model's performance through curriculum learning (refer to Section V-C), multiple pairs of augmented views are generated for each circuit, creating a series of datasets with progressively increasing learning difficulty. More specifically, our rewriting algorithm is applied to each truth table circuit using different replacement ratio thresholds, denoted as T , ranging from 0.1 to 0.5. This results in a pair of augmented graphs for each threshold. Overall, we build a large pretraining dataset that consists of more than two million netlists, covering a wide variety of equivalent functional transformations. This helps FGNN2 learn the basic knowledge about logic functionality, which is transferable across different netlists and different tasks. For further details regarding the generated pretraining dataset, please refer to TABLE I.

Algorithm 1 Structurally Diverse Logic Rewriting

```

1: procedure REWRITE( $AIG, T, Library$ )
2:   for each  $N \in topo(AIG)$  do
3:     for each  $C \in Cut_4(N)$  do
4:        $V \leftarrow Lookup(Library, F(C));$ 
5:        $BestS \leftarrow NULL, BestGain \leftarrow -1;$ 
6:       for  $S \in V$  do
7:         if  $Gain(S) > BestGain$  then
8:            $BestGain \leftarrow Gain(S);$ 
9:            $BestC \leftarrow C;$ 
10:           $BestS \leftarrow S;$ 
11:        end if
12:      end for
13:    end for
14:    if  $BestS \neq NULL$  then
15:       $Replace(AIG, N, BestC, BestS);$ 
16:       $O(BestS) \leftarrow O(BestS) + 1;$ 
17:       $N_r \leftarrow N_r + |C|;$ 
18:    end if
19:    if  $N_r \geq T \times |AIG|$  then
20:      break;
21:    end if
22:  end for
23: end procedure

```

V. CIRCUIT CONTRASTIVE LEARNING SCHEME

A. Customized Contrastive Loss

As depicted in Fig. 4, our proposed circuit contrastive scheme adheres to a standard paradigm, where the objective is to maximize the agreement between different views of the same circuit. Specifically, we employ mini-batch gradient descent to train our circuit contrastive learning scheme, where a mini-batch consisting of N pairs of augmented circuits is

randomly sampled at each epoch. It is important to note that each sampled pair (c'_1, c'_2) represents the same Boolean functionality, and thus it is regarded as a positive pair. Following the sampling strategy described in [37], we define the **negative samples** for any positive pair (c'_1, c'_2) as the remaining $(2N - 2)$ circuits within the mini-batch, as their functionality is different from that of (c'_1, c'_2) . A normalized temperature-scaled cross-entropy loss (NT-Xent) [38] can then be applied to maximize the consistency between positive pairs compared with negative samples, which can be written as follows:

$$\mathcal{L} = \frac{1}{N} \sum_{n=1}^N l_{1,2}(n) + l_{2,1}(n), \quad (4)$$

where $l_{1,2}(n) + l_{2,1}(n)$ gives the loss for the positive pair of the n -th circuit, and $l_{i,j}$ is defined as Equation (5), where $\mathbf{h}_{n,i}$ denotes the embedding of the i -th augmentation of n -th circuit, $\sim(\mathbf{h}_{n,i}, \mathbf{h}_{n,j}) = \frac{\mathbf{h}_{n,i}^\top \mathbf{h}_{n,j}}{\|\mathbf{h}_{n,i}\| \|\mathbf{h}_{n,j}\|}$ is the cosine similarity between the two embeddings $\mathbf{h}_{n,i}$ and $\mathbf{h}_{n,j}$ and τ is a temperature parameter.

The intuition of the original contrastive loss function (see Equation (5)) is to enhance the similarity between positive pairs while reducing the similarity between negative pairs. This is achieved by placing the similarity score between the positive samples in the numerator and the similarity scores between the negative samples in the denominator of the loss function. By minimizing the loss function, we can bring the embeddings of the positive samples closer (increased similarity) while pushing the embeddings of the negative samples further apart (decreased similarity). However, this loss function was originally designed for the field of computer vision (CV) and may not perfectly align with our problem due to the inherent differences between the two fields. In computer vision, it is challenging to measure the relative distance to the target sample between two negative samples, such as determining which dog is more similar to a cat. As a result, the vanilla contrastive loss function (Equation (5)) assumes that all negative samples should be treated equally, assigning the same weight (all one) to each negative pair $(h_{u,i}, h_{n,i})$.

However, this assumption does not hold when it comes to logic functionality learning. Consider the following **question**: Is the distance between a 3-input OR gate and a 2-input OR gate the same as the distance between a 3-input OR gate and a 3-input AND gate? Intuitively, the answer is no, as the first two gates implement a similar Boolean function, while the latter two gates are logically distinct. The above observation emphasizes the need to develop a method for accurately measuring such differences in logic functionality.

In this paper, we propose a method for measuring logic differences in circuits by utilizing the truth table, which serves as a straightforward and comprehensive representation of a circuit's functionality. As shown in Fig. 5, each row of a truth table depicts the behavior (output value) of a circuit given a specific input pattern. The input patterns are organized in rows, typically using lexicographic ordering. Meanwhile, each output column in the table represents the logic functionality of the corresponding output bit, referred to as the bit's truth vector. As previously mentioned, our pre-training dataset

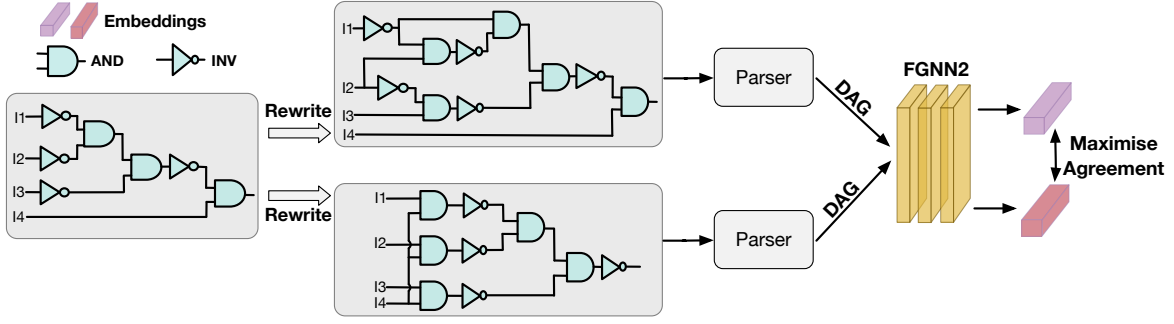
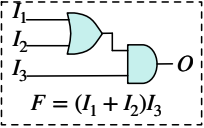


Fig. 4 Overview of our circuit contrastive learning framework. Refer to Section V for details.

$$l_{i,j}(n) = -\log \frac{\exp(\sim(h_{n,i}, h_{n,j})/\tau)}{\sum_{u=1, u \neq n}^N \exp(\sim(h_{n,i}, h_{u,i})/\tau) + \sum_{u=1}^N \exp(\sim(h_{n,i}, h_{u,j})/\tau)}, \quad (5)$$

I_1	I_2	I_3	O	I_1	I_3	I_2	O
0	0	0	0	0	0	0	0
0	0	1	0	0	0	1	0
0	1	0	0	0	1	0	0
0	1	1	1	0	1	1	1
1	0	0	0	1	0	0	0
1	0	1	1	1	0	1	0
1	1	0	0	1	1	0	1
1	1	1	1	1	1	1	1
TT_a				TT_b			

I_1
 I_2
 I_3

 $F = (I_1 + I_2)I_3$

$v_a^t = [0, 0, 0, 1, 0, 1, 0, 1]$
 $v_b^t = [0, 0, 0, 1, 0, 0, 1, 1]$
 $v_a^f = v_b^f = [0, 0, \frac{1}{4}, \frac{1}{8}]$

Fig. 5 The change of the input order can lead to different truth vectors. Nevertheless, our proposed functional vector remains the same. Here TT is for truth table, F refers to the Boolean function, v^t is for the vanilla truth vector, and v^f is for our functional vector.

exclusively consists of circuits with a single output, i.e., with only one truth vector. Consequently, the truth vector of each pre-training circuit serves as an effective representation of its functionality.

Based on the above observations, there comes a straightforward way to measure the logical difference between two one-output circuits, i.e., based on the distance between the truth vectors. However, there are several practical challenges in applying such a measurement, including:

- 1) **Capturing the relationship between circuits with different input widths:** To fully comprehend the logic functionality, it is essential to consider the relationship between circuits with mismatched input widths. While measuring the distance between truth vectors of the same length (i.e., same input width) is straightforward, it becomes significantly more challenging when dealing with truth vectors of different lengths.
- 2) **Handling the variance of input signal order:** In practical scenarios, there is often no prior knowledge regarding the order of a circuit's input signals. Consequently, the method used for functional distance measurement should be invariant to changes in input order. However, the

conventional truth vector is sensitive to the order of input signals, making it an inadequate choice. For instance, altering the input order from $I_1 I_2 I_3$ to $I_1 I_3 I_2$ in Fig. 5 does not alter the logic functionality, but it results in a different truth vector.

To address the first challenge, we leverage the utilization of don't care conditions within truth tables. In particular, to measure the distance between two circuits with different input widths, we expand the small one by introducing don't care input terms, as illustrated in Fig. 6. These don't-care input terms can be eliminated through logical simplification, thereby not affecting the output value. Consequently, the addition of these terms does not alter the functionality of the circuit. In practice, we enhance our pre-training dataset by randomly extending some generated synthetic circuits. This approach aids in capturing the relationship between circuits with different input widths and further strengthens the effectiveness of our proposed method.

To address the second challenge, we introduce a novel form of a functional vector that remains invariant to changes in input order. To begin with, we divide the input patterns into groups based on the positive ratio, which refers to the number of ones in each pattern. For instance, in the case of 3-input truth tables, there are a total of $2^3 = 8$ input patterns, which are divided into four groups: $G_0 = \{000\}$, $G_1 = \{001, 010, 100\}$, $G_2 = \{011, 110, 101\}$, and $G_3 = \{111\}$. Next, we encode the functional vector for each truth table circuit by calculating the positive probability of the output signal conditioned on each group. This can be expressed as follows:

$$v^f = [P(O=1|G_i) * P(G_i) \text{ for } i = 0, 1, 2, 3] \quad (6)$$

And the functional distance between two one-output circuits a and b can be computed as Equation (7), where \sim denotes cosine similarity.

$$d_{ab} = 1 - \sim(v_a^f, v_b^f) \quad (7)$$

As shown in Fig. 5, this approach enables us to generate the same vector for functionally equivalent circuits, regardless

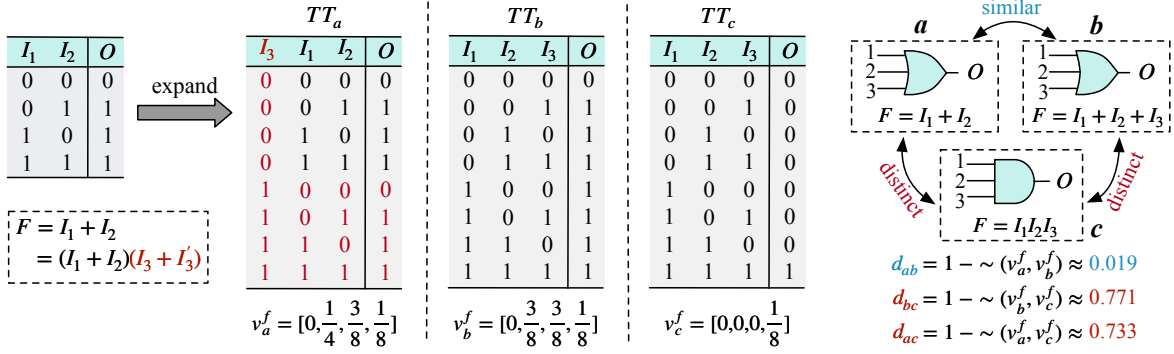


Fig. 6 An example illustrating the functional distance between OR_2 , OR_3 , and AND_3 gates, which are represented by cases a, b, and c, respectively. Here we also show how to extend a 2-input circuit (OR_2 gate) to a 3-input one by adding a don't care input I_3 .

of the input order. Furthermore, it allows us to provide a definitive answer to the question raised earlier in this section: the distance between an OR_2 gate and an OR_3 gate is smaller than the distance between an OR_3 gate and an AND_3 gate. As shown in Fig. 6, the functional distance between OR_2 and OR_3 is 0.019, while the distance between OR_3 and AND_3 is 0.771. It is evident that our proposed functional vector effectively captures the relative functional distance between these three circuits.

Based on the above functional distance, we propose a novel contrastive loss function that takes into account the relative distance between different negative samples and the same target sample. This is achieved by assigning the functional distance as a weight to each negative pair, which reflects their relative functional distance. Generally, negative pairs with larger function distances are assigned larger weights. As a result, negative samples with more distinct functionality will be pushed further away from the target sample in the embedding space. The formal expression of our customized contrastive loss function is given by Equation (8).

The experimental results presented in Section VI have demonstrated the effectiveness of our proposed functional vector and customized contrastive loss function. However, it is crucial to recognize that our approach does have certain limitations, providing opportunities for future enhancements. A primary drawback of our proposed functional vector is its exclusive focus on the conditional probability of generating an output of 1. As a result, our method may produce identical functional vectors for two circuits that display similar yet different functionalities. Despite this limitation, the advantages of our new contrastive loss function outweigh the potential drawbacks. Notably, even if two circuits possess the same vector but exhibit different functionalities, they are still regarded as a negative pair. This implies that our designed loss function Equation (8) does not attempt to draw their embeddings closer. Therefore, the negative impact of this limitation is expected to be minor. Moving forward, we plan to enhance our method by designing a more robust functional vector and contrastive loss function.

B. Functional Graph Neural Network

While Graph Neural Networks (GNNs) have offered a powerful approach for representation learning in graphs, it is still crucial to carefully determine the GNN architecture that aligns with our pre-training scheme. In light of the characteristics of circuits, customization of GNNs should address two key aspects: (1) ensuring knowledge transferability across various circuits; and (2) the capacity to learn logic functionality. To tackle these challenges, we implement a customized GNN architecture, namely Functional Graph Neural Network (FGNN).

To satisfy the first requirement of ensuring robust generalization capability, we opt to learn the functionality of AND gates and Inverters independently within our FGNN architecture. This approach is inspired by relational GCN [39] and is rooted in the understanding that logic gates, being the fundamental building blocks of circuits, inherently reflect the underlying semantics of circuits and maintain consistent functionality across different designs. Specifically, our FGNN model consists of two learnable message aggregators, namely the AND aggregator and the INV aggregator. The AND aggregator is designed to generate the embeddings for AND gates, which first take the average of the predecessor embeddings of an AND gate and then perform MLP and activation. On the other hand, the INV aggregator is designed for inverters, which performs MLP and activation on the inverter's input embedding. By learning these essential gate functions, our model effectively captures the generic knowledge that is shared among various circuits.

Moreover, to effectively address the second requirement of learning logic functionality, our FGNN employs an asynchronous message-passing scheme [8]. This scheme is specifically designed to emulate the logical computation process inherent in gate-level circuits. Through this method, the functionality of a circuit can be naturally integrated into the embeddings of the Primary Output (PO) nodes. To reduce the graph depth and alleviate the over-smoothing issue, we move the inverter nodes onto the edges, as depicted by the diamonds on edges in Fig. 7. For any target node v , the message passing scheme starts from the Primary Inputs (PIs) of v 's fanin-cone and all the way to v . Throughout this process, a node remains

$$l_{i,j}(n) = -\log \frac{\exp(\sim (\mathbf{h}_{n,i}, \mathbf{h}_{n,j})/\tau)}{\exp(\sim (\mathbf{h}_{n,i}, \mathbf{h}_{n,j})/\tau) + \sum_{u=1}^N \sum_{k=i,j} \exp(d_{ni,uk} \sim (\mathbf{h}_{n,i}, \mathbf{h}_{u,k})/\tau)}, \quad (8)$$

inactive until representations for all its predecessors have been computed. Once a node is activated, FGNN aggregates the received messages (representations) from its predecessors to construct its representation, which is then propagated to all its successors. For Primary Output (PO) nodes with an inverted outgoing edge, we apply the inverter aggregator to its representation. The above message passing scheme is illustrated in Fig. 7.

To sum up, we use an asynchronous relational GCN to embed logic functionality. Formally, the message aggregating scheme of a node v can be stated as follows:

$$\begin{aligned} \mathbf{m}_v^i &= \mathcal{A}^{inv}(\{h_u : u \in \mathcal{P}^i(v)\}); \\ &= \sigma(MLP^{inv}(\{h_u | u \in \mathcal{P}^i(v)\})); \\ \mathbf{h}_v &= \mathcal{A}^{and}(\mathbf{m}_v^i, \{h_u : u \in \mathcal{P}^n(v)\}); \\ &= \sigma(MLP^{and}(f(\mathbf{m}_v^i, \{h_u : u \in \mathcal{P}^n(v)\}))); \end{aligned} \quad (9)$$

where $\mathcal{P}^i(v)$ denotes the set of predecessor nodes of v connected by an inverter edge, and $\mathcal{P}^n(v)$ denotes the set of predecessor nodes of v connected by a non-inverter edge. \mathcal{A}^{and} and \mathcal{A}^{inv} are the learnable aggregators for AND gates and inverters, respectively. MLP^{and} and MLP^{inv} are Multilayer Perceptrons. σ is the leaky-Relu activation function, and f is the combination function, which is instantiated as the mean operator. The initial message for each primary input is an all-one vector.

The node embeddings learned by FGNN can be directly fed into a classifier/regression model to handle local-level tasks like link prediction or node classification. For global scenarios, e.g., circuit classification, we first select a set of representative nodes V^r from the target graph and then use a readout function READOUT (e.g., mean, sum, etc.) to combine the selected nodes' representations into a single global-level representation \mathbf{h}_{global} , as described in Equation (10),

$$\mathbf{h}_{global} = \text{READOUT}(\{\mathbf{h}_u : u \in V^r\}) \quad (10)$$

In practice, we select the Primary Outputs (POs) of a target circuit as the representative vertices and use a concatenation of mean and max functions to read out.

C. Curriculum Learning

We have incorporated a curriculum learning [40] scheme into our pre-training procedure to address the challenges associated with learning difficulty. The idea behind curriculum learning is to gradually expose the model to increasingly complex or informative examples, starting with simpler ones, to help it learn more effectively [41]. Instead of randomly shuffling the training data, curriculum learning arranges the examples based on their difficulty or relevance to the learning task. The curriculum can be designed based on various factors, such as the complexity of the samples, their similarity to the

target task, or their potential to provide useful insights for the model.

By presenting examples in a curated order, curriculum learning aims to guide the model's learning process, allowing it to gradually build foundational knowledge before tackling more challenging instances. This approach helps prevent the model from becoming overwhelmed by difficult examples early on and has the potential to improve its generalization and convergence speed.

In the context of learning logic functionality, our curriculum is carefully designed with two key considerations: (1) the complexity of the circuit (measured by the number of inputs) and (2) the topological similarity between pairs of augmented circuits (measured by the replacement ratio). In particular, our curriculum is initialized with circuits with 4 inputs and a replacement ratio of 10%. Within each curriculum, the samples are shuffled at the beginning of each epoch, ensuring that each circuit is exposed to a sufficient number of negative samples. As the training loss reaches a plateau, i.e., no improvement in consecutive epochs, we move on to the next curriculum, which involves circuits with an increased number of inputs or a larger replacement ratio. For more detailed information regarding our curriculum, please refer to TABLE I and Section VI-A.

VI. EXPERIMENTS

A. Experimental Settings

We implemented our circuit representation learning framework with DGL [42], a graph learning library based on PyTorch [43]. FGNN2 is pre-trained and fine-tuned on a Linux machine with 48 Intel Xeon Silver 4212 cores (2.20GHz), 1 GeForce RTX 3090 Ti GPU, and 32 GB of main memory.

As discussed in Section V-C, we have employed the curriculum learning technique to guide the pre-training process. Specifically, we denote a dataset composed of circuits with m inputs and a replacement ratio of $r\%$ during augmentation as (m,r) . The pre-training procedure starts with learning on easy cases denoted as $(4,10)$. Once the training loss reaches a plateau, i.e., no improvement in consecutive n epochs, we progress to more challenging cases with an increased number of inputs or a larger replacement ratio. The entire pre-training dataset sequence is set as follows: $(4,10)$, $(4,20)$, $(4,30)$, $(4,50)$, $(5,10)$, $(5,20)$, $(5,30)$, $(5,40)$, $(5,50)$, $(6,10)$, $(6,20)$, $(6,30)$, $(6,40)$, $(6,50)$, $(7,10)$, $(7,20)$, $(7,30)$, $(7,40)$, and $(7,50)$. Throughout the pre-training process, we set the batch size to 512, the cumulative epoch threshold to $n = 5$, and the temperature parameter to $\tau = 0.05$.

Regarding the hyper-parameters of our GNN model, we set the dimension of the output representations to 256. The MLP layers, denoted as MLP^{and} and MLP^{inv} , are instantiated as 3-layer Multiplayer Perceptrons (MLPs) with a hidden

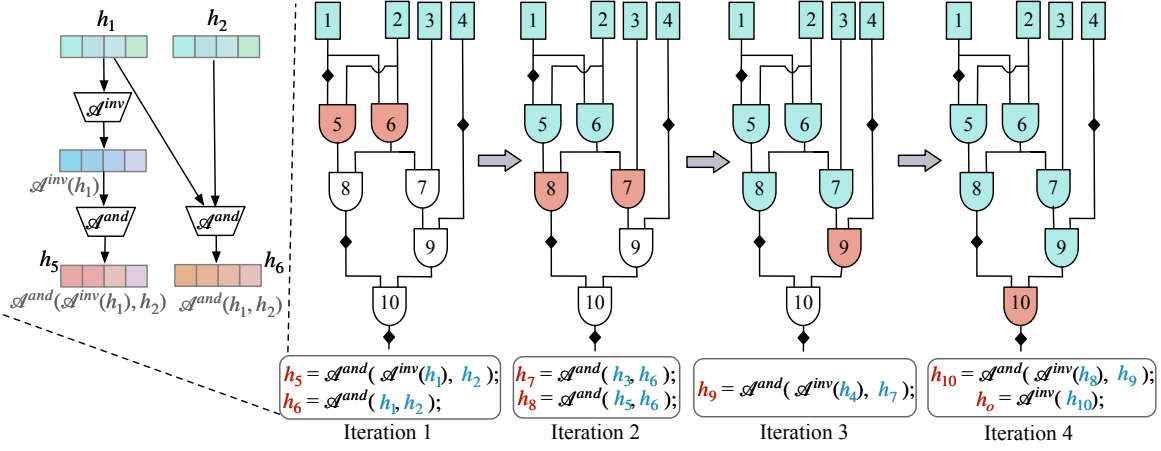


Fig. 7 An illustration of the message passing process of FGNN. Square vertices are primary inputs initializing with all-one vectors, and the edges with a diamond are inverted edges. Different states of vertices are indicated by fill colors: white means inactive, red means being updated at the current iteration and blue means having been updated in previous iterations. \mathcal{A}^{and} and \mathcal{A}^{inv} are different learnable aggregators depicted by Equation (9).

dimension of 128 and the leaky ReLU activation function. The negative slope for all the leaky ReLU functions is set to 0.01.

To evaluate the performance of our proposed framework, we conducted experiments on two different downstream circuit tasks, covering both local and global scenarios. We fed the target circuits from these tasks into our pre-trained FGNN model to generate node representations. Subsequently, we constructed a graph-level representation for each circuit using the method described in Equation 10. Finally, the node/graph representations were input into a classifier (MLP) to fine-tune the model and make predictions.

B. Evaluation on Circuit Classification

For the global scenario, we choose the circuit classification task to evaluate our method, which involves the identification of circuits based on their respective functions. Specifically, our focus lies in accurately classifying arithmetic circuits, including adder, subtractor, multiplier, and divider. This task presents challenges as it requires grouping circuits with similar functionalities but varying structures, such as the Brent-Kung adder and Sklansky adder. Additionally, distinguishing circuits with similar structures but different functionalities, such as adders and subtractors, adds further complexity.

The downstream training dataset consists of randomly generated arithmetic circuits in the form of word-level Verilog. These circuits are then synthesized into gate-level circuits using Synopsys Design Compiler, employing different constraints to incorporate diverse architectures for each arithmetic block. Meanwhile, the downstream validation and test datasets consist of arithmetic circuits with unseen architectures, allowing us to assess the generalization capability of our model. Specifically, we use adder/multiplier designs from the work of [44], which possess distinct architectures compared to the training circuits. Similarly, the test subtractor designs are generated using different constraints from the ones used for training, resulting in disparate architectures. These circuits

TABLE II Statistics of the dataset for circuit classification, including adder, subtractor, multiplier, and divider. We try to avoid involving similar architectures used for training in the test dataset.

Module	Train		Validate / Test	
	architectures	#	architectures	#
Adder	Brent-Kung, Cond-Sum, Hybrid, Koggle-Stone, Ling, Sklansky	450	Block Carry Look-head, Carry Look-head, Carry Select, Carry-skip, Ripple-Carry	100 + 300
Subtractor	Hybrid, Koggle-Stone, Ling	250	Brent-Kung, Cond-Sum, Sklansky	50 + 150
Multiplier	Array, Booth-Encoding	550	Wallace, Dadda, Overturned-stairs, (4,2) compressor, (7,3) counter, Redundant binary addition	150 + 500
Divider	Array	250	Array	50 + 200
Total	/	1500	/	350 + 1150

TABLE III The size and the average training time of the models for the global task (ratio=1.3). Here we only list the training time on the largest ratio. For other ratios, the approximate time can be derived proportionally.

	GIN [17]	EV-CNN [6]	DVAE [24]	RelGCN [39]	DeepGate2 [12]	FGNN2
#parameters (k)	230	25	130	656	758	164
Train time (h)	0.7	0.4	0.5	0.9	2.8	0.6

are further converted into the AIG form utilizing ABC. The circuits operate on word lengths ranging from 8 to 32 bits, with the number of gates ranging from hundreds to thousands. Further details regarding the dataset statistics are presented in II. For evaluation purposes, we utilize accuracy as the performance metric.

We reimplemented several representative prior works [6], [12], [17], [24], [39] as the baseline methods for comparison. These works have covered both structure-based methods [6],

TABLE IV Summary of performance on circuit classification in terms of accuracy. The second column gives the ratio of the training data size to the testing data size. Our proposed FGNN2 framework achieves the **best** performance on all the cases and suffers from no degradation when the training data scale is reduced.

Case	Ratio	GIN [17]	EV-CNN [6]	DVAE [24]	RelGCN [39]	Deepgate2 [12]	FGNN [11]	Ours
1	1.3	0.762	0.904	0.913	0.929	0.997	0.975	1.000
2	1	0.745	0.896	0.902	0.921	0.963	0.962	1.000
3	0.7	0.737	0.884	0.895	0.912	0.945	0.960	1.000
4	0.5	0.730	0.877	0.885	0.900	0.933	0.951	1.000
5	0.3	0.725	0.859	0.871	0.883	0.928	0.945	1.000

[17], [24], [39], and the state-of-the-art (SOTA) functionality-aware methods [11], [12]. For the structure-based methods, we train their models from scratch on the downstream circuit classification dataset, utilizing their official implementations. Regarding our previous work FGNN [11], we report the performance as presented in the original paper. As for DeepGate2 [12], we employ their open-source pre-trained model and fine-tune it on the downstream dataset. To ensure fairness, we use the same fine-tuning settings for both FGNN2 and DeepGate2, including the architecture of the MLP-based classifier, the readout function, and other relevant parameters. For all models, we train or fine-tune them on the downstream dataset for a total of 100 epochs. The size and training time of the models are listed in TABLE III.

To thoroughly evaluate the models' generalization ability and robustness in the face of limited training data, we maintain fixed validation and testing datasets while training and fine-tuning the models using datasets of varying sizes. The results are summarized in TABLE IV. It is evident from the table that our proposed framework consistently outperforms the baseline methods in all scenarios.

Remarkably, our FGNN2 model exhibits no degradation in performance when the training data size is reduced. This result demonstrates the efficacy of our proposed pretraining framework in extracting the universal knowledge about logic functionality. In comparison, the performance of the baseline models declines sharply as the training dataset size decreases. It is worth mentioning that our FGNN2 model achieves high accuracy even with a training data size that is only 30% of the testing data size, resulting in a performance gain of 5.5% compared to the second-best method [11].

C. Evaluation on Sub-circuit Identification

In this part, we assess our proposed framework in a local scenario, arithmetic block identification. Arithmetic blocks refer to the fundamental components within a circuit that perform arithmetic operations, e.g., integer addition. The boundaries of these blocks are defined by the input/output wires interacting with external circuits [8]. Our task is to accurately recognize the output boundaries of adders within a large circuit design. We adopt the experimental setup from the previous work [8] and measure performance using recall and F1-score metrics. To evaluate the scalability of our proposed method, we use some large-scale RISC-V CPU designs [45] to form the dataset, including Rocket, a 5-stage in-order scalar core, and Berkeley Out-of-Order (BOOM) Core, an out-of-order superscalar RV64G core [46]. We use BOOM

TABLE V Statistics of the dataset for sub-circuit identification with 6 different types of adders.

Architecture	<i>Rocket (test)</i>		<i>BOOM (train)</i>	
	#nodes	#edges	#nodes	#edges
Brent-Kung	109558	145794	782483	1099295
Cond-sum	116655	152770	778840	1093248
Hybrid	116894	152923	783713	1099169
Kogge-Stone	116703	152313	780463	1094997
Ling	118529	155999	785812	1104923
Sklansky	116348	152490	784356	1101919

TABLE VI The size and the average training time of the models for the local task (ratio=6/6). Here we only list the training time on the largest ratio. For other ratios, the approximate time can be derived proportionally.

	EV-CNN [6]	SAGE [18]	ABGNN [8]	RelGCN [39]	FGNN2
#parameters (k)	25	160	115	656	164
Train time (h)	1.5	3.1	1.9	3.7	2.3

as the training set and leave Rocket for validation and testing with a splitting ratio of 1 : 9. The designs are synthesized with Synopsys Design Compiler using the SAED 32/28nm Digital Standard Cell Library. Various settings of design constraints are applied to generate netlists with different adder architectures. Subsequently, we convert the generated netlists into the AIG form. Detailed information about the dataset is presented in TABLE V.

We compare our method with representative baselines that focus on generating node-level representations. These baselines include both structure-based methods [6], [8], [18], [39] and functionality-aware methods [11]. Similarly, we train the structure-based models from scratch on the downstream dataset, using their official implementations. For all models, we train or fine-tune them on the downstream dataset for a total of 100 epochs. The size and training time of the models are listed in TABLE VI.

To assess the generalization ability of the models, we employ a fixed testing/validation dataset that includes all six different adder architectures. In contrast, our training dataset only covers a subset of the adder architectures (e.g., ratio = 1/6 indicates the utilization of one out of six different architectures). We conduct the experiments 10 times for each ratio $\frac{k}{6}$, shuffling the training dataset and using the first k architectures each time. The average performance is reported in TABLE VII, which demonstrates the superiority

TABLE VII Performance of different models on adder output boundary prediction in terms of recall and F1-score. Best results are emphasized with **boldface**. Our proposed FGNN2 framework outperforms other models in all the test cases.

Case	Ratio	EV-CNN [6]		GraphSage [18]		ABGNN [8]		RelGCN [39]		FGNN [11]		FGNN2	
		Recall	F1-Score	Recall	F1-Score	Recall	F1-Score	Recall	F1-Score	Recall	F1-Score	Recall	F1-Score
1	1/6	0.602	0.575	0.643	0.656	0.657	0.682	0.768	0.742	0.734	0.753	0.864	0.872
2	2/6	0.612	0.605	0.758	0.757	0.734	0.74	0.794	0.799	0.857	0.839	0.924	0.916
3	3/6	0.633	0.615	0.854	0.865	0.877	0.881	0.860	0.882	0.940	0.937	0.949	0.942
4	4/6	0.662	0.637	0.883	0.889	0.921	0.91	0.884	0.911	0.954	0.947	0.956	0.948
5	5/6	0.738	0.648	0.905	0.898	0.927	0.922	0.929	0.931	0.966	0.951	0.968	0.951
6	6/6	0.768	0.655	0.919	0.917	0.945	0.941	0.947	0.948	0.969	0.957	0.969	0.959

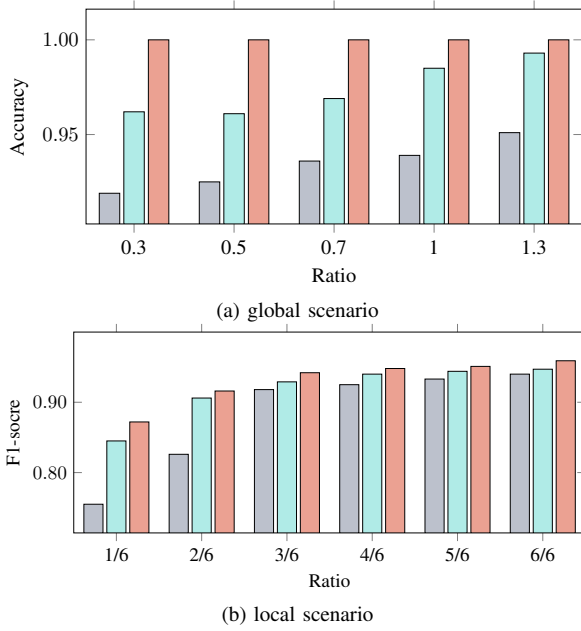


Fig. 8 Performance comparison between FGNN2 without pre-training (grey), FGNN2 pre-trained with vanilla loss (blue), and FGNN2 pre-trained with our customized loss (orange) on circuit classification Fig. 8(a) and sub-circuit identification Fig. 8(b).

of our methods compared to several state-of-the-art circuit representation learning methods. It is worth noting that the functionality-aware method FGNN [11] outperforms previous structure-based work on all the cases, highlighting the crucial role of logic functionality in generating powerful circuit representations. However, FGNN exhibits significant performance degradation when generalizing to unseen data. For instance, its performance drops by 11.2% when only two of the adder structures are included in the training dataset (case 2). In contrast, our FGNN2 displays much more stable performance, experiencing smaller degradation in all cases (e.g., a decrease of only 4.5% in case 2). These results indicate that our proposed FGNN2 can effectively identify the output boundary of adders with unseen architectures. In summary, this demonstrates the effectiveness of our proposed contrastive framework in extracting high-level prior knowledge of circuits.

D. Ablation Study

We also conduct ablation studies to validate the efficacy of our proposed contrastive loss function. Specifically, we estab-

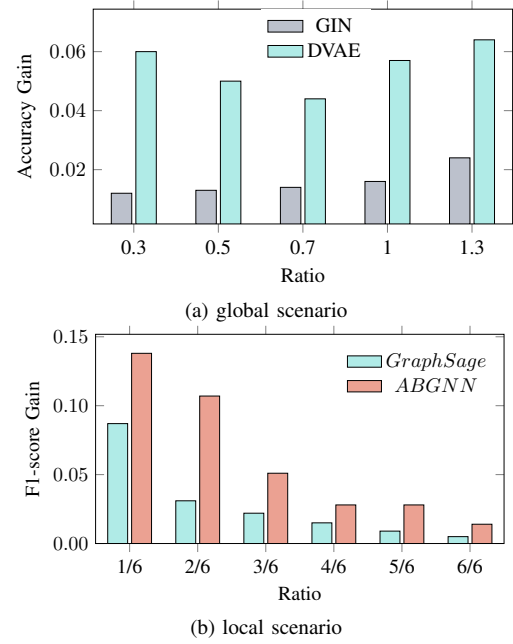


Fig. 9 The performance gain of our pre-training scheme when combined with representative baseline methods.

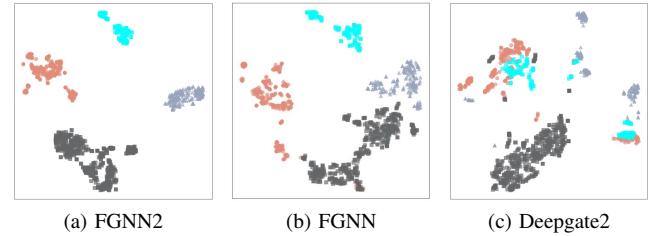


Fig. 10 Visualization of the embedding space of our pre-trained FGNN2 and the SOTA works DeepGate2 and FGNN (all without fine-tuning), where orange, blue, black, and grey represent adders, subtractor, multipliers, and dividers.

lished two baseline models: 1) $FGNN2^i$: an FGNN2 model that has not undergone pre-training, i.e., the learnable weights are randomly initialized; and 2) $FGNN2^v$: an FGNN2 model that has been pre-trained with a standard contrastive loss (see Equation (5)). These two baseline models are then compared with the fully developed version, $FGNN2^c$, which has been pre-trained with our customized contrastive loss function (see Equation (8)). It is important to note that the second baseline $FGNN2^v$ is pre-trained with the same dataset and the same data augmentation scheme as $FGNN2^c$. The only

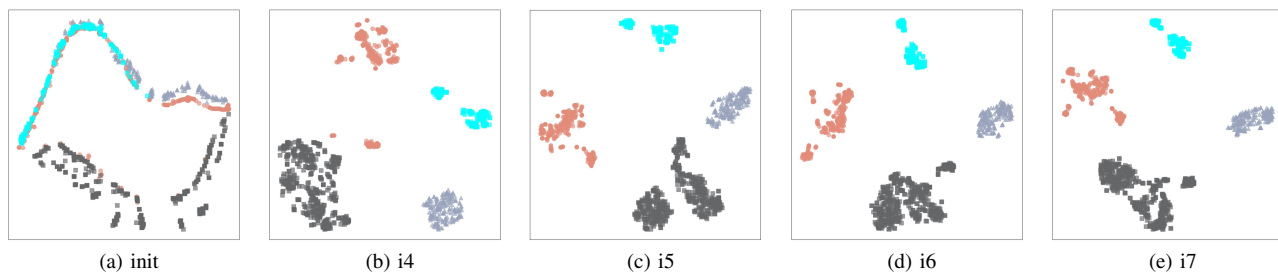


Fig. 11 Visualization of the embedding space of our pre-trained model at different curriculum stages, where i4 refers to the curriculum datasets consisting of 4-input circuits, so as i5, i6, and i7.

difference between $FGNN2^v$ and $FGNN2^c$ lies in the used loss function to guide the pre-training process. As depicted in Fig. 8, the results demonstrate that $FGNN2^v$ outperforms $FGNN2^i$ in both downstream tasks, thus highlighting the effectiveness of our contrastive-based pre-training approach. Moreover, this outcome underscores the significance of logic functionality in generating robust circuit representations. Furthermore, $FGNN2^c$ exhibits additional improvements over $FGNN2^v$, providing evidence of the efficacy of our customized contrastive loss function. This finding also suggests that capturing the relative functional distance between circuits contributes to a better understanding of logic functionality.

Additionally, we carried out experiments to further explore the impact of our proposed pre-training scheme by integrating it with four representative baseline models, including GIN [17], DVAE [24], GraphSage [18] and ABGNN [8]. We present the performance gain of our pre-training scheme on these methods in Fig. 9. As can be observed, asynchronous GNNs that are designed for directed graphs such as DVAE and ABGNN, are more compatible with our proposed pre-training scheme than synchronous GNNs like GIN and GraphSage. This could be attributed to the fact that asynchronous GNNs propagate messages in the topological order, which mimics the process of logic computation. Consequently, they can better capture the Boolean functionality of the pre-training circuits.

E. Visualization

To further demonstrate the effectiveness of our proposed FGNN2, we present visualization results for the circuit classification task. Specifically, we employ the t-Distributed Stochastic Neighbor Embedding (t-SNE) algorithm [47], a widely used technique for dimensionality reduction and visualization of high-dimensional data. The primary objective of t-SNE is to map high-dimensional data points to a lower-dimensional space while preserving their pairwise similarities, which aligns well with our requirements. For the t-SNE algorithm, we set the target dimension to 2, the perplexity to 50, and the maximum number of optimization iterations to 1000.

We first compare the embedding spaces of our pre-trained FGNN2 with the pre-trained models of two state-of-the-art (SOTA) baselines, FGNN and DeepGate2. It is important to note that these pre-trained models have not undergone any fine-tuning for the downstream task, meaning they have not been exposed to the arithmetic circuits. From Fig. 10,

it can be seen that the arithmetic circuits are already well separated in the embedding space of FGNN2, even without any fine-tuning. In contrast, the embedding spaces of FGNN and DeepGate2 exhibit a mixture of different arithmetic circuits. For example, DeepGate2 struggles to distinguish between adders (colored orange) and subtractors (colored blue). This could be attributed to the simple loss function used in DeepGate2, which limits the exposure of each circuit to an adequate number of negative samples. In summary, these results further demonstrate the superiority of our proposed FGNN2 in learning circuit functionality compared to the SOTA works FGNN and DeepGate2.

Furthermore, we conduct visualizations to illustrate the process of our curriculum learning scheme. Fig. 11(a) shows the embedding space of our FGNN2 model without pre-training, i.e., with randomly initialized weights, where the arithmetic circuits disperse throughout the embedding space. From Figs. 11(b) to 11(e), it can be seen that the arithmetic blocks tend to be clustered and separated based on their functionality, with the progression of the curriculum learning.

F. Effect of Curriculum Learning

We also designed experiments to reveal the impact of curriculum learning. Specifically, we run our pre-training scheme under three settings: 1) Directly on the hard-to-learn dataset (7,10) without pre-trained on easier datasets, namely '(7,10) wo CL'; 2) Directly on the easy-to-learn dataset (4,10), namely '(4,10)'; and 3) On the hard-to-learn dataset (7,10) after pre-training on (4,10), namely '(7,10) w CL'. We run all these settings for 100 epochs and report the loss during training in Fig. 12. As seen in Fig. 12, applying the pre-training scheme directly to complex cases, i.e., the dataset (7,10), results in a failure to converge, as indicated by the blue line. This could be due to the high learning difficulty associated with complex designs. In contrast, the pre-training scheme converges well on the easy dataset (4,10), as depicted by the orange line. Moreover, the pre-training scheme also converges well on (7,10) if the model has been pre-trained on easier cases, as shown by the black line.

VII. CONCLUSION

Learning feasible representations from raw gate-level circuits is critical for applying machine learning techniques to EDA. In this work, we propose a novel pre-training

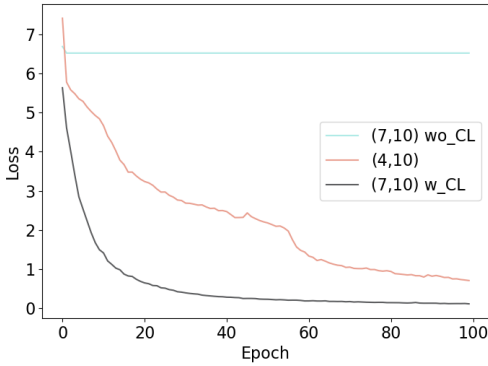


Fig. 12 The pre-training loss under three different settings.

framework for generating functionality-aware circuit representations, based on a customized circuit contrastive learning scheme. To construct a comprehensive synthetic pre-training dataset, we employ a customized circuit augmentation scheme and leverage the logic synthesis tool ABC. We also introduce a novel contrastive loss function that effectively captures the relative functional distance between circuits. Additionally, we incorporate a specialized graph neural network to further enhance the performance of our pre-training framework. Experimental results on two distinct downstream tasks verified the framework's effectiveness. In the future, we plan to broaden our research scope to include much larger-scale Boolean networks in our pre-training dataset, and to apply our pre-training model to a wider range of EDA tasks, e.g., SAT problem.

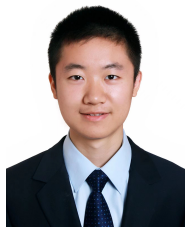
REFERENCES

- [1] H. Geng, Y. Ma, Q. Xu, J. Miao, S. Roy, and B. Yu, "High-speed adder design space exploration via graph neural processes," *IEEE TCAD*, 2021.
- [2] G. Huang, J. Hu, Y. He, J. Liu, M. Ma, Z. Shen, J. Wu, Y. Xu, H. Zhang, K. Zhong *et al.*, "Machine learning for electronic design automation: A survey," *ACM TODAES*, vol. 26, no. 5, pp. 1–46, 2021.
- [3] Z. Wang, S. Liu, Y. Pu, S. Chen, T.-Y. Ho, and B. Yu, "Restructure-tolerant timing prediction via multimodal fusion," in *Proc. DAC*. IEEE, 2023, pp. 1–6.
- [4] S. Liu, Z. Wang, F. Liu, Y. Lin, B. Yu, and M. Wong, "Concurrent sign-off timing optimization via deep steiner points refinement," in *Proc. DAC*. IEEE, 2023, pp. 1–6.
- [5] S. Zheng, L. Zou, P. Xu, S. Liu, B. Yu, and M. Wong, "Lay-Net: Grafting Netlist Knowledge on Layout-Based Congestion Prediction," in *Proc. ICCAD*. IEEE, 2023, pp. 1–9.
- [6] A. Fayyazi, S. Shababi, P. Nuzzo, S. Nazarian, and M. Pedram, "Deep learning-based circuit recognition using sparse mapping and level-dependent decaying sum circuit representations," in *Proc. DATE*, 2019, pp. 638–641.
- [7] Y. Ma, H. Ren, B. Khailany, H. Sikka, L. Luo, K. Natarajan, and B. Yu, "High performance graph convolutional networks with applications in testability analysis," in *Proc. DAC*, 2019, pp. 1–6.
- [8] Z. He, Z. Wang, C. Bai, H. Yang, and B. Yu, "Graph learning-based arithmetic block identification," in *Proc. ICCAD*, 2021.
- [9] M. Li, Z. Shi, Q. Lai, S. Khan, S. Cai, and Q. Xu, "DeepSAT: An eda-driven learning framework for sat," *arXiv preprint arXiv:2205.13745*, 2022.
- [10] K. Zhu, H. Chen, W. J. Turner, G. F. Kokai, P.-H. Wei, D. Z. Pan, and H. Ren, "TAG: Learning circuit spatial embedding from layouts," in *Proc. ICCAD*, 2022, pp. 1–9.
- [11] Z. Wang, C. Bai, Z. He, G. Zhang, Q. Xu, T.-Y. Ho, B. Yu, and Y. Huang, "Functionality matters in netlist representation learning," in *Proc. DAC*, 2022, pp. 61–66.
- [12] Z. Shi, H. Pan, S. Khan, M. Li, Y. Liu, J. Huang, H.-L. Zhen, M. Yuan, Z. Chu, and Q. Xu, "DeepGate2: Functionality-aware circuit representation learning," in *Proc. ICCAD*, 2023, pp. 1–9.
- [13] Z. Wang, C. Bai, Z. He, G. Zhang, Q. Xu, T.-Y. Ho, B. Yu, and Y. Huang, "Fgnn2," 2024. [Online]. Available: <https://github.com/ZeyayW/FGNN2>
- [14] Z. Wu, S. Pan, F. Chen, G. Long, C. Zhang, and S. Y. Philip, "A comprehensive survey on graph neural networks," *IEEE Transactions on Neural Networks and Learning Systems*, vol. 32, no. 1, pp. 4–24, 2020.
- [15] Y. Ma, Z. He, W. Li, L. Zhang, and B. Yu, "Understanding graphs in EDA: From shallow to deep learning," in *Proc. ISPD*, 2020, pp. 119–126.
- [16] X. Zheng, Y. Liu, S. Pan, M. Zhang, D. Jin, and P. S. Yu, "Graph neural networks for graphs with heterophily: A survey," *arXiv preprint arXiv:2202.07082*, 2022.
- [17] K. Xu, W. Hu, J. Leskovec, and S. Jegelka, "How powerful are graph neural networks?" *Proc. ICLR*, 2018.
- [18] W. Hamilton, Z. Ying, and J. Leskovec, "Inductive representation learning on large graphs," in *Proc. NIPS*, 2017, pp. 1024–1034.
- [19] P. Veličković, G. Cucurull, A. Casanova, A. Romero, P. Lio, and Y. Bengio, "Graph attention networks," *Proc. ICLR*, 2018.
- [20] M. Zhang and Y. Chen, "Link prediction based on graph neural networks," *Proc. NIPS*, vol. 31, pp. 5165–5175, 2018.
- [21] Y. Gong, Y. Zhu, L. Duan, Q. Liu, Z. Guan, F. Sun, W. Ou, and K. Q. Zhu, "Exact-K Recommendation via Maximal Clique Optimization," *Proc. KDD*, pp. 617–626, 2019.
- [22] T. Chen, Q. Sun, C. Zhan, C. Liu, H. Yu, and B. Yu, "Deep h-gcn: Fast analog IC aging-induced degradation estimation," *IEEE TCAD*, 2021.
- [23] V. Thost and J. Chen, "Directed acyclic graph neural networks," *arXiv preprint arXiv:2101.07965*, 01 2021.
- [24] M. Zhang, S. Jiang, Z. Cui, R. Garnett, and Y. Chen, "D-VAE: A variational autoencoder for directed acyclic graphs," *Proc. NIPS*, 2019.
- [25] P. Bachman, R. D. Hjelm, and W. Buchwalter, "Learning Representations by Maximizing Mutual Information Across Views," *Proc. NIPS*, vol. 32, pp. 15 535–15 545, 2019.
- [26] T. Chen, S. Kornblith, M. Norouzi, and G. Hinton, "A simple framework for contrastive learning of visual representations," in *Proc. ICML*, 2020, pp. 1597–1607.
- [27] X. Wang, R. Zhang, C. Shen, T. Kong, and L. Li, "Dense contrastive learning for self-supervised visual pre-training," in *Proc. CVPR*, 2021, pp. 3024–3033.
- [28] E. Xie, J. Ding, W. Wang, X. Zhan, H. Xu, P. Sun, Z. Li, and P. Luo, "Detco: Unsupervised contrastive learning for object detection," in *Proc. CVPR*, 2021, pp. 8392–8401.
- [29] B. Poole, S. Ozair, A. Van Den Oord, A. Alemi, and G. Tucker, "On variational bounds of mutual information," in *Proc. ICML*, 2019, pp. 5171–5180.
- [30] P. Veličković, W. Fedus, W. L. Hamilton, P. Liò, Y. Bengio, and R. D. Hjelm, "Deep graph infomax," *Proc. ICLR*, vol. 2, no. 3, p. 4, 2019.
- [31] Y. You, T. Chen, Y. Sui, T. Chen, Z. Wang, and Y. Shen, "Graph contrastive learning with augmentations," *Proc. NIPS*, vol. 33, pp. 5812–5823, 2020.
- [32] Y. Zhu, Y. Xu, F. Yu, Q. Liu, S. Wu, and L. Wang, "Graph contrastive learning with adaptive augmentation," in *The Web Conference*, 2021, pp. 2069–2080.
- [33] Y. Tian, C. Sun, B. Poole, D. Krishnan, C. Schmid, and P. Isola, "What makes for good views for contrastive learning?" *Proc. NIPS*, vol. 33, pp. 6827–6839, 2020.
- [34] T. Xiao, X. Wang, A. A. Efros, and T. Darrell, "What should not be contrastive in contrastive learning," *arXiv preprint arXiv:2008.05659*, 2020.
- [35] R. Brayton and A. Mishchenko, "ABC: An academic industrial-strength verification tool," in *Proc. CAV*. Springer, 2010, pp. 24–40.
- [36] P. Pan and C.-C. Lin, "A new retiming-based technology mapping algorithm for LUT-based FPGAs," in *Proc. FPGA*, 1998, pp. 35–42.
- [37] T. Chen, Y. Sun, Y. Shi, and L. Hong, "On sampling strategies for neural network-based collaborative filtering," in *Proc. KDD*, 2017, pp. 767–776.
- [38] K. Sohn, "Improved deep metric learning with multi-class n-pair loss objective," in *Proc. NIPS*, 2016, pp. 1857–1865.
- [39] M. Schlichtkrull, T. N. Kipf, P. Bloem, R. Van Den Berg, I. Titov, and M. Welling, "Modeling relational data with graph convolutional networks," in *15th Extended Semantic Web Conference (ESWC)*, 2018, pp. 593–607.
- [40] Y. Bengio, J. Louradour, R. Collobert, and J. Weston, "Curriculum learning," in *Proc. ICML*, 2009, pp. 41–48.

- [41] X. Wang, Y. Chen, and W. Zhu, "A survey on curriculum learning," *IEEE TPAMI*, vol. 44, no. 9, pp. 4555–4576, 2021.
- [42] M. Wang, L. Yu, D. Zheng, Q. Gan, Y. Gai, Z. Ye, M. Li, J. Zhou, Q. Huang, C. Ma *et al.*, "Deep graph library: Towards efficient and scalable deep learning on graphs," *arXiv preprint arXiv:1909.01315*, 2019.
- [43] A. Paszke, S. Gross, S. Chintala, G. Chanan, E. Yang, Z. DeVito, Z. Lin, A. Desmaison, L. Antiga, and A. Lerer, "Automatic differentiation in pytorch," 2017.
- [44] N. Homma, Y. Watanabe, T. Aoki, and T. Higuchi, "Formal design of arithmetic circuits based on arithmetic description language," *IEICE Trans. Fundamentals*, vol. 89, no. 12, pp. 3500–3509, 2006.
- [45] A. Amid, D. Biancolin, A. Gonzalez, D. Grubb, S. Karandikar, H. Liew, A. Magyar, H. Mao, A. Ou, N. Pemberton *et al.*, "Chipyard: Integrated Design, Simulation, and Implementation Framework for Custom SoCs," *IEEE Micro*, vol. 40, no. 4, pp. 10–21, 2020.
- [46] K. Asanovic, R. Avizienis, J. Bachrach, S. Beamer, D. Biancolin, C. Celio, H. Cook, D. Dabbelt, J. Hauser, A. Izraelevitz *et al.*, "The rocket chip generator," *EECS Department, University of California, Berkeley, Tech. Rep. UCB/EECS-2016-17*, 2016.
- [47] L. Van der Maaten and G. Hinton, "Visualizing data using t-SNE," *Journal of Machine Learning Research*, vol. 9, no. 11, 2008.



Ziyi Wang received his B.S. degree from the Department of Computer Science and Technology, Fudan University in 2021. He is currently pursuing his Ph.D. degree at the Department of Computer Science and Engineering, The Chinese University of Hong Kong. His research interests include graph learning applications in electronic design automation (EDA) and logic synthesis.



Chen Bai received the B.E. degree in software engineering from the University of Electronic Science and Technology of China, Chengdu, China, in 2020. He is currently pursuing the Ph.D. degree with the Department of Computer Science and Engineering, The Chinese University of Hong Kong, Hong Kong. His research interests include computer architecture and electronic design automation. Mr. Bai received the William J. McCalla Best Paper Award from ICCAD 2021 and the Best Paper Award Nomination from ISPD 2024.



Zhuolun He received his Ph.D. degree from the Chinese University of Hong Kong (CUHK) in 2023, and his B.S. degree in computer science and engineering from Peking University in 2017. He is currently a postdoctoral fellow at the Department of Computer Science and Engineering, CUHK.



Guangliang Zhang is a principal ASIC engineer of Hisilicon and has rich experience in front-end design, DFT design, and back-end design. His current focus is on the competitiveness of digital chips.



Qiang Xu (S'02-M'05-SM'21) received his B.E. and M.E. degrees from Beijing University of Posts and Telecommunications, and his Ph.D. degree from McMaster University. He is a Professor of Computer Science and Engineering, The Chinese University of Hong Kong. His research interests include electronic design automation, trusted computing and representation learning. He has published 170+ papers in related fields and received a number of best paper awards and/or best paper nominations. He is currently serving as an associate editor of IEEE Transactions on Computer-Aided Design and Systems and Integration, the VLSI Journal. He was an associate editor of IEEE Design and Test. He has served as a program committee member of 50+ international conferences on electronic design automation, computer security and artificial intelligence.



Tsung-Yi Ho (F'24) is a Professor in the Department of Computer Science and Engineering, the Chinese University of Hong Kong (CUHK). He received his Ph.D. in Electrical Engineering from National Taiwan University in 2005. His research interests include several areas of computing and emerging technologies, especially in design automation of microfluidic biochips. He was a recipient of the Best Paper Award at the IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems in 2015. Currently, he serves as the VP Conferences of IEEE CEDA, and the Executive Committee of ASP-DAC and ICCAD. He is a Distinguished Member of ACM and a Fellow of IEEE.



Yu Huang is Semiconductor Scientist of Huawei, EDA Chief Architect, and EDA Lab director of HiSilicon. Before Joining HiSilicon, he was Sr. Key Expert of Mentor Graphics. His research interests include VLSI SoC testing, ATPG, compression, diagnosis, yield analysis, machine learning, and AI chips. He got his Ph.D. in electrical and computer engineering from the University of Iowa in 2002, USA. He has about 90 patents and published more than 150 papers in leading IEEE Journals, conferences, and workshops. He is a senior member of the IEEE. He has served as a technical program committee member for DAC, ITC, VTS, ATS, ETS, ASPDAC, NATW and many other conferences and workshops in the testing area. He is also an adjunct professor at the School of Microelectronics, Fudan University, Xidian University, and Nanjing University of Posts and Telecommunications China.



Bei Yu (M'15-SM'22) received the Ph.D. degree from The University of Texas at Austin in 2014. He is currently an Associate Professor in the Department of Computer Science and Engineering, The Chinese University of Hong Kong. He has served as TPC Chair of ACM/IEEE Workshop on Machine Learning for CAD, and in many journal editorial boards and conference committees. He is the Editor of IEEE TCCPS Newsletter. He received ten Best Paper Awards from IEEE TSM 2022, DATE 2022, ICCAD 2021 & 2013, ASPDAC 2021 & 2012, ICTAI 2019, Integration, the VLSI Journal in 2018, ISPD 2017, SPIE Advanced Lithography Conference 2016, and six ICCAD/ISPD contest awards.