Functionality Matters in Netlist Representation Learning

Ziyi Wang
CUHK

Chen Bai
CUHK

Zhuolun He
CUHK

Guangliang Zhang
HiSilicon

Qiang Xu
CUHK

Tsung-Yi Ho
CUHK

Bei Yu
CUHK

Yu Huang
HiSilicon

Abstract

Learning feasible representation from raw gate-level netlists is essential for incorporating machine learning techniques in logic synthesis, physical design, or verification. Existing message-passing-based graph learning methodologies focus merely on graph topology while overlooking gate functionality, which often fails to capture underlying semantic, thus limiting their generalizability. To address the concern, we propose a novel netlist representation learning framework that utilizes a contrastive scheme to acquire generic functional knowledge from netlists effectively. We also propose a customized graph neural network (GNN) architecture that learns a set of independent aggregators to better cooperate with the above framework. Comprehensive experiments on multiple complex real-world designs demonstrate that our proposed solution significantly outperforms state-of-the-art netlist feature learning flows.

1 Introduction

As machine learning (ML) techniques develop rapidly, there is a surge in incorporating ML in electronic design automation (EDA) [1–3]. Most existing works follow a representation learning paradigm consisting of two steps: first, learn low-dimensional representations from the high-dimensional raw data and then conduct classification or regression based on the learned representations. The learned representations play a dominant role in improving model performance. In this work, we focus on representation learning for electronic circuit netlists. This is non-trivial as a netlist contains circuit components, which might vary largely in structures and connectivity. Therefore, it is worth designing subtle representation learning methodologies dedicated to netlists.

Early netlist representation construction methods focus on the structural information of netlists. Structural information mainly includes the topology of circuit components. A representative art [4] developed a shape hashing technique to group wires with similar local topology into words, where a sequence representation named shape is proposed. Specifically, a target wire's *shape* is produced by serializing gate and wire types along depth-first-search (DFS) traversal of its fan-in cone. However, the generated representation is highly related to the traversing order, which is stochastic. As a consequence, wires with isomorphic fan-in structures may be pushed apart in the representation space. Besides, the information contained in the traversal sequence is relatively shallow, leading to insufficient understanding of the netlists.

DAC '22, July 10–14, 2022, San Francisco, CA, USA
© 2022 Association for Computing Machinery.
ACM ISBN 978-1-4503-9142-9/22/07…$15.00
https://doi.org/10.1145/3489517.3530602

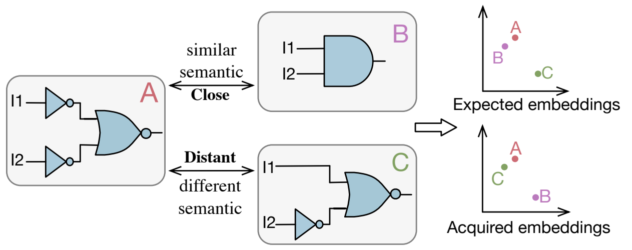

Figure 1: Illustration of the drawback of existing structure-based netlist representation learning methods.

In recent years, the fast-growing deep neural network techniques have shown great power in netlist representation learning. A compact representation termed level-dependent decaying sum (LDDS) existence vector (EV) is introduced in [5] to embed a circuit node with its neighbors. A fixed number of EVs are selected to satisfy the fixed-input-size requirement of convolutional neural networks. Ma et al. [1] propose an iterative process to insert observation points into gate-level netlists based on the node representations learned by a graph neural network. [6] develop a graph learning-based solution to extract desired logic components from a flattened netlist, where a novel graph neural network customized for directed acyclic graph (DAG) is proposed to generate gate representations. The embeddings are then fed into a classifier to predict the boundaries of desired components. While generating more powerful representations, the above machine learning-driven methods can likewise be categorized as structural ones since they take merely topological information into consideration.

Though existing structure-based netlist representation learning methods have achieved state-of-the-art performance in many netlist-level tasks, we argue that they are far from good enough. The main point is that these methods ignore the boolean functionality, which plays a dominant role in understanding the semantics of netlists. Figure 1 gives two examples to illustrate the drawbacks of existing structure-based netlist representation learning methods. Netlists A, B, and C are three different netlists. A and B implement the same function, sharing similar semantics, which means they should be close in the representation space. However, existing methods would push their representations apart since they are topologically disparate. On the other hand, A and C implement different functions, thus having different semantics, and are expected to be distant in the representation space. Nevertheless, their representations would be pulled together by existing methods based on their similar structures.

To address the above concerns, we develop a novel contrastive learning (CL)-based netlist representation learning framework to extract the basic logic functionality of netlists, which is universal and transferable across different designs. We further propose a customized graph neural network architecture to better cooperate with the above framework. The proposed framework aims at encoding functional information of netlists, independently of specific structural patterns, thus improving the capability of generalizing to unseen designs.

Our major contributions are summarized as follows:

- For the first time, to the best of our knowledge, we present a contrastive learning-based pre-training framework customized for gate-level circuits, extracting universal knowledge of logic functionality.
- We design a novel GNN architecture for circuit representation learning that encodes the basic functional information of gate-level netlists.
- We conduct comprehensive experiments on several complex real-world designs, which confirms the effectiveness of our proposed framework compared with state-of-the-art netlist representation learning arts.

2 Preliminaries

2.1 Graph Neural Network

Graph neural networks (GNNs) [7–9] have emerged as a promising approach for analyzing graph-structured data in recent years. They follow an iterative neighborhood aggregation scheme to capture the structural information within nodes' neighborhoods. Let $G = \langle \mathcal{V}, \mathcal{E} \rangle$ denotes a graph, where $\mathcal{V} = \{v_1, v_2, \cdots, v_n\}$ 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

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

where $\boldsymbol{h}_v^{(k)}$ 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.

Various GNNs [10–12] have been proposed, achieving state-of-the-art performance in related graph learning tasks. Notably, there emerges growing interest in a unique graph type, directed acyclic graph (DAG)[13, 14]. DAGs are widely applied to model many real-world data, including gate-level netlists. To generate better global-level embeddings for DAGs, [13] construct an impressive GNN architecture driven by the partial order induced by DAG. Besides, [14] propose an asynchronous message passing scheme to encode computation graphs and further develop a variational autoencoder for DAGs, D-VAE. However, these methods can only be applied to handle structural information of DAGs while omitting the underlying semantics of target graphs, which is vital for generating better representations.

2.2 Graph Contrastive Learning

Contrastive learning (CL) is employed as a new paradigm to pre-train the model to improve the modeling performance on downstream tasks. The main idea of contrastive learning is to capture statistical dependencies of interest and those that do not by separating positive samples from negative samples in the n-dimensional embedding space R^n as much as possible. The goal of CL is to learn an encoder $f : x \rightarrow \boldsymbol{e}, \boldsymbol{e} \in \mathbb{R}^n$ that for any sample x:

$$score(f(x), f(x^+)) >> score(f(x), f(x^-)). \tag{2}$$

Here x^+ refers to positive samples that are similar or equal to x, x^- refers to negative samples that are different from x, and $score(\boldsymbol{e}_1, \boldsymbol{e}_2)$ measures the similarity (distance) between embedding $\boldsymbol{e}_1$ and $\boldsymbol{e}_2$.

Due to its outstanding performance, contrastive learning achieves great success in the computer vision domain [15, 16]. Theoretical analyses shed light on the reasons behind their success [17]: objectives used in contrastive methods can be seen as maximizing a lower bound of mutual information between the input data and the output representations.

In recent years, many researchers have been focusing on extending the contrastive methods to handle graph data. DGI [18] embeds high-order global contextual features into node representations by maximizing mutual information between global and local embeddings. You *et al.* [19] build multiple views of a graph by incorporating several perturbations, e.g., edge dropping, node dropping, feature masking, etc. GCA [20] is proposed to explore graph data augmentation.

However, most existing 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 consistent with the semantics in many situations, e.g., two structurally different netlists may share the same function (e.g., ripple-carry adder and carry look-ahead adder). Besides, even a small perturbation may totally change the semantics of a graph (e.g., replacing an XOR gate with an OR gate may change a circuit's function). Therefore, existing methods fail to guarantee the consistency between the augmented view and the original input since they randomly introduce structural perturbations (e.g., random edge/node dropping), which may change the underlying semantics.

3 Problem Formulation

We first introduce the gate-level netlist and then give the problem formulation. The gate-level netlist of an electric circuit consists of a list of gate-level circuit components, e.g., `AND` gates and interconnects (wires) between them. Gate-level netlists are generated by converting a description of circuit behavior at register transfer level (RTL) into design implementation in logic gates.

A gate-level netlist can be formulated as a DAG with vertices denoting circuit components and edges representing wires. Based on the gate-level netlist, our problem can be formulated.

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

4 Netlist Representation Learning Framework

Before introducing algorithmic details, we briefly overview our proposed netlist representation learning flow, as shown in Figure 2. The whole flow can be summarized as three steps:

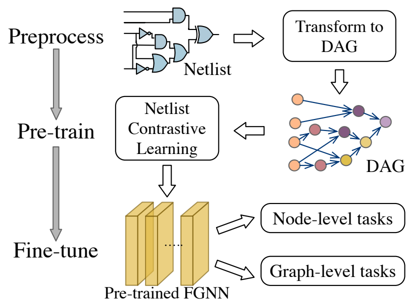

Figure 2: Our proposed netlist representation learning flow

1. Preprocessing: Firstly, the input gate-level netlist is transformed to a DAG. Each vertex corresponds to a gate in the netlist, and directed edges represent the interconnection between gates, where the source vertices denote the driven gates.

2. Pre-training: Secondly, functionality graph neural network (FGNN) (Section 4.2) is dedicated designed for learning functionality of gate-level netlists. The pre-training of FGNN is guided by the proposed gate-level netlist contrastive scheme (Section 4.1).

3. Fine-tuning: Finally, pre-trained FGNN is equipped with a classifier and fine-tuned to adapt downstream tasks.

4.1 Netlist Contrastive Scheme

To extract the prior knowledge of netlists' basic logic functionality, we develop a novel netlist contrastive learning (NCL) scheme, as shown in Figure 4. Our proposed scheme follows a standard contrastive learning paradigm, where the model seeks to maximize the agreement of different views (constructed through data augmentation) of the same item. Researches have shown that the success of contrastive learning lies in the assumption that important information is shared between different views [21, 22]. It indicates that the semantics of augmented views should be maintained following the original input.

While generating augmented views without causing semantic changes is relatively natural for images (e.g., translation, scale) [16], it is not explicit for graph data. The primary difficulty here is that the semantics of graphs are not apparent under many situations, and sometimes even a tiny perturbation may completely alter its meaning. To resolve the above difficulty and extend the contrastive learning scheme to handle graphs, it is critical to explore a customized data augmentation scheme that introduces perturbations without changing the original semantics.

When it comes to a gate-level netlist, its semantics can be denoted by its function. Given a gate-level netlist, we can treat it as a computation graph and describe its function through a boolean expression, whose basic operators are the logic gates, as demonstrated in Figure 3. Therefore, based on the observation of equivalent boolean transformation, we carefully design a data augmentation scheme to maximize the mutual information between functionally similar netlists. Specifically, for a given netlist c, we conduct the augmentation by replacing a randomly picked sub-netlist in c with equivalent Boolean transformations (described in Figure 4), resulting in a functionally constant but topologically different netlist c'. This procedure guarantees the semantic (functionality) consistency between the augmented netlist c' and the original netlist c, making the contrastive objective clear and explicit. By applying the above augmentation

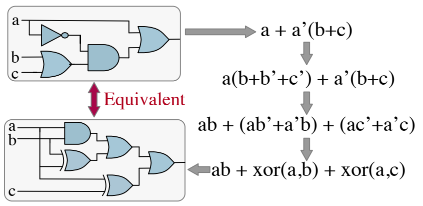

Figure 3: Illustration of Boolean Equivalence

scheme to netlist c twice, we obtain two augmented netlists (c_1', c_2') as a positive pair .

We apply mini-batch gradient descent to train our netlist contrastive learning scheme. Each time a mini-batch of N netlists are randomly sampled and processed through the above functionality-constant augmentation, resulting in $2N$ augmented netlists. Following the sampling strategy in [23], we define the negative samples for any positive pair (c_1', c_2') as the other $(2N-2)$ netlists within the mini-batch. A normalized temperature-scaled cross-entropy loss (NT-Xent) [24] is then applied to maximize the consistency between positive pairs compared with negative samples, which is formulated as follows:

$$\mathcal{L} = \frac{1}{N}\sum_{n=1}^{N} l_{1,2}(n) + l_{2,1}(n), \tag{3}$$

where $l_{1,2}(n) + l_{2,1}(n)$ gives the loss for the positive pair of the n-th netlist, and $l_{i,j}$ is defined as:

$$l_{i,j}(n) = -\log \frac{\exp(\sim(\boldsymbol{h}_{n,i}, \boldsymbol{h}_{n,j})/\tau)}{\sum\limits_{u=1,u\neq n}^{N} \exp(\sim(\boldsymbol{h}_{n,i}, \boldsymbol{h}_{u,i})/\tau) + \sum\limits_{u=1}^{N} \exp(\sim(\boldsymbol{h}_{n,i}, \boldsymbol{h}_{u,j})/\tau)}, \tag{4}$$

where $\boldsymbol{h}_{n,i}$ denotes the embedding of the i-th augmentation of n-th netlist, $\sim(\boldsymbol{h}_{n,i}, \boldsymbol{h}_{n,j}) = \boldsymbol{h}_{n,i}^{\top}\boldsymbol{h}_{n,j}/\|\boldsymbol{h}_{n,i}\|\|\boldsymbol{h}_{n,j}\|$ is the cosine similarity between the two embeddings $\boldsymbol{h}_{n,i}$ and $\boldsymbol{h}_{n,j}$ and τ is a temperature parameter.

4.2 Customized Graph Neural Network: FGNN

As mentioned in Section 2.1, though enabling a powerful representation learning paradigm for graphs, existing GNNs fail to capture the underlying semantic of netlists. Hence, designing customized architecture that adapts to netlist is still crucial to achieving better performance. This part discusses how to design a novel graph neural network architecture that complements our netlist contrastive learning scheme.

Regarding the unique properties of gate-level netlists, GNN customization should take two aspects into account: (1) how to learn logic functionality and (2) how to guarantee knowledge transferability between different netlists. We find that most existing GNNs do not fit well since they learn to encode topological information (e.g., node connectivity) instead of logic functionality. Consequently, the learned knowledge is highly related to the training graphs' topology, leading to poor generalization ability.

To overcome the above drawbacks, we propose a novel GNN architecture that targets learning the basic logical functionality of netlists, namely functional graph neural network (FGNN). Specifically, instead of learning a shared aggregator for all the nodes, FGNN learns a set of independent aggregators (functions), one for each gate type (e.g., `AND, XOR`…). The insight behind this is that, as the most fundamental building component of netlists, the logic gates'

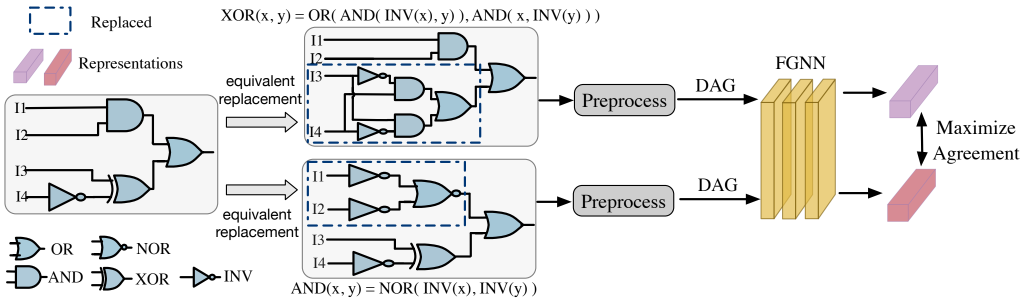

Figure 4: Overview of our Netlist Contrastive representation learning framework. Refer to Section 4.1 for details.

functionalities naturally reflect the underlying semantics of netlists and keep constant across different designs. By learning the essential gate functions, our proposed model manages to capture the generic knowledge shared between different netlists. In practice, we learn 8 basic gate (cell) functions including `AND`, `OR`, `INV`, `MAJ`, `MUX`, `NAND`, `NOR` and `XOR`.

Motivated by ABGNN [6], our proposed FGNN follows an asynchronous message passing scheme. For any target vertex v_i, the message passing scheme starts from the Primary Inputs (PIs) of v_i's fanin-cone and all the way to v_i. During the procedure, a vertex stays inactive until all its predecessors' representations have been computed. When a vertex is activated, FGNN aggregates the messages (representations) received from its predecessors to construct its own representation and then sends the newly-built representation to all its successors. Our work differs from ABGNN [6] since we use independent aggregators for vertices of different types. The above message passing scheme is illustrated in Figure 5.

Formally, given a target vertex v, the aggregation scheme of the k-th iteration of a depth-δ FGNN can be described as follows:

$$\begin{aligned}\boldsymbol{h}^{(k)}_{\{\mathrm{i}:\mathcal{D}(\mathrm{i},\mathrm{v})=\delta-\mathrm{k}\}} &= \mathcal{A}_{\{\mathrm{g}:\mathcal{T}(\mathrm{i})=\mathrm{g}\}}(\{\boldsymbol{h}^{(k-1)}_u : u \in \mathcal{N}(i)\}) \\ &= \sigma(W_g \cdot f_g(\{\boldsymbol{h}^{(k-1)}_u : u \in \mathcal{N}(i)\})),\end{aligned} \tag{5}$$

where $\mathcal{D}(i, v)$ is the distance between vertices i and v in the graph, $\mathcal{N}(i)$ is the set of direct neighbors of vertex i, $\mathcal{T}(i)$ gives the type of vertex i, $\mathcal{A}_g$ is the aggregator for vertex type g, σ is an activation function, W_g is the weight matrix for $\mathcal{A}_g$ and f_g is the reduce function for $\mathcal{A}_g$. The initial message for each vertex i is an all-one vector. The red part in the formulation emphasizes the difference between our work and ABGNN [6].

The vertex representations 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., graph classification, we first select a set of representative vertices V from the target graph and then use a readout function READOUT (e.g., mean, sum, etc.) to combine the selected vertices' representations into a single global-level representation $\boldsymbol{h}_{global}$, as described in Equation (6),

$$\boldsymbol{h}_{global} = \mathrm{READOUT}(\{\boldsymbol{h}_u : u \in V\}) \tag{6}$$

In practice, we select the Primary Outputs (POs) of a target netlist as the representative vertices and use a mean function to readout.

4.3 Curriculum Learning

We further introduce a curriculum learning scheme to guide the pre-training procedure. We first train the model on a small number

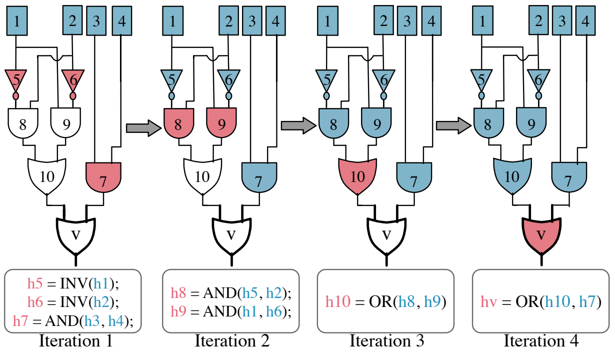

Figure 5: An illustration of the representation generating procedure. The target vertex v is emphasized with bold outlines. Square vertices are primary inputs initializing with all-one representations. Different states of vertices are indicated by fill colors: blank means inactive, red means being computed at current iteration and blue means having been computed in previous iterations. INV, AND, OR are different aggregators.

of easy cases and then train on successively more complex cases with increased batch size.

Two aspects determine the perplexity of a positive pair: (1) netlist complexity (depth/number of PIs) and (2) topological similarity between the two composed netlists (measured by the times of functionality-constant replacements). Simpler cases and cases with similar positive pairs are regarded as easier ones.

The curriculum is initialized with netlists with 4 PIs. Each time the training performance plateaus below a threshold loss, the perplexity of the cases is increased by either adding number of PIs or introducing more variance (applying more cell replacements) during data augmentation, up to a maximum of K PIs and M times replacements. In our experiments, we set $K = 8$ and $M = 4$.

5 Experiments

We implemented the netlist representation learning framework with DGL [5], a graph learning library based on PyTorch. FGNN is pre-trained/fine-tuned on a Linux machine with 48 Intel Xeon Silver 4212 cores (2.20GHz), 1 GeForce RTX 2080 Ti GPU, and a 32 GB main memory.

As mentioned in Section 4.3, we utilize the curriculum learning technique to guide the pre-training procedure. Specifically, we denote a dataset composed of netlists with m Primary Inputs (PIs) and up to n times cell replacements during augmentation as (m,n). The

Table 1: Performance of different models on adder output boundary prediction in terms of recall and F1-score. Best results are emphasized with boldface. Our proposed FGNN + NCL framework outperforms other models in all the test cases.

Case	Ratio	EV-CNN [5]		GraphSage [8]		ABGNN [6]		FGNN		FGNN + NCL	
		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	0.684	0.715	**0.734**	**0.753**
2	2/6	0.612	0.605	0.758	0.757	0.734	0.74	0.784	0.788	**0.857**	**0.839**
3	3/6	0.633	0.615	0.854	0.865	0.877	0.881	0.916	0.914	**0.940**	**0.937**
4	4/6	0.662	0.637	0.883	0.889	0.921	0.917	0.931	0.933	**0.954**	**0.947**
5	5/6	0.738	0.648	0.905	0.898	0.927	0.922	0.952	0.944	**0.966**	**0.951**
6	6/6	0.768	0.655	0.919	0.917	0.945	0.941	0.963	0.952	**0.969**	**0.957**

Table 2: Statistics of the dataset for sub-netlist identification with 6 different types of adders.

Architecture	*Rocket (test)*		*BOOM (train)*	
	#gates	#wires	#gates	#wires
Brent-Kung	24340	58124	139526	366280
Cond-sum	24737	57708	138358	360455
Hybrid	25491	60287	141319	369622
Kogge-Stone	24540	57726	139005	361962
Ling	26179	62864	143903	378354
Sklansky	25208	59567	141093	369774

pre-training procedure starts from learning on easy cases denoted as (4,1). Each time the loss goes continuously lower than a threshold t, we change to more challenging cases with mores PIs or increased augmenting variance (more replacements). The training dataset sequence is set as (4,1), (5,1), (5,2), (6,1) (6,2), (6,3), (7,1), (7,2), (7,3), (8,1), (8,2), (8,3), (8,4). During training, we set the loss threshold $t = 0.1$ and temperature parameter $\tau = 0.065$. The datasets we use are composed of 50,000 5-PI netlists, 75,000 6-PI netlists, 100,000 7-PI netlists, and 100,000 8-PI netlists.

The performance of our proposed framework is evaluated on two different downstream netlist tasks covering both local and global scenarios. We feed the downstream target netlists into our pre-trained FGNN to generate node representations and further construct a global-level representation for each netlist as described in Equation (6). The representations are then fed into a classifier (Multilayer Perceptron, MLP) to fine-tune the model and make predictions.

5.1 Evaluation on Sub-netlist Identification

We first assess our proposed framework on a local scenario, arithmetic block identification. Arithmetic blocks are the building blocks within a netlist that perform certain arithmetic operations (e.g., integer addition), whose boundaries are defined as the input/output wires interacting with external circuits [6]. In general, our task is to recognize the boundaries of target arithmetic blocks from a large netlist design. Here we focus on identifying the output boundaries of adders, following the same experimental setting as [6], where the performance is measured in terms of recall and F1-score. The detail of the dataset we use is shown in Table 2.

We consider representative baselines that are dedicated to generating node-level representations, covering the following three categories: (1) CNN-based method [5], (2) general GNN-based method [8], and (3) customized GNN-based method that adapts to DAGs [6]. All these baselines are trained end-to-end, and we report their performance based on their official implementations.

To evaluate the models' generalization ability, we fix the testing/validation dataset containing all the six different adder architectures (shown in table 2) and train/fine-tune the models with data that involves only part of the adder architectures. (e.g., $ratio = 1/6$ means using one out of six different types). The result in Table 1 demonstrates the superiority of our methods compared with several State-of-the-Art netlist representation learning methods. Our method stands out in all the cases and achieves 2.4% $\sim$ 12.3% recall gain and 1.2% $\sim$ 10.5% F1-Score improvement compared with the second-best solution [6]. Additionally, we can see that GNN-based methods [6, 8] significantly outperform previous CNN-based work on all the cases and achieve relatively good performance when testing on data similar to the training dataset, confirming the power of GNN in netlist representation learning. However, they are subjected to sharp performance degradation when generalizing to unseen data. For instance, their performance drops by around 8% when half of the test adder structures are not involved in the training dataset (case 3). In contrast, the performance of our FGNN is much more stable, suffering from pretty minor degradation on all the cases (e.g., decreased by only 4% on case 3). Moreover, when combined with our novel netlist contrastive learning flow, our model's generalization ability is further enhanced, achieving comparable results on case 3 with other methods' best performance. This result shows the effectiveness of our proposed contrastive framework in extracting high-level prior knowledge of netlists.

5.2 Evaluation on Nelist Classification

For the global scenario, we choose the netlist classification task, which targets distinguishing between netlists of different functions. Here we focus on classifying arithmetic netlists, including adder, subtractor, multiplier, and divider. The training netlists are randomly generated in word-level Verilog and synthesized into gate-level circuits by Synopsys Design Compiler. Different constraints are used during synthesis to involve diverse architectures for each module. The validate/test dataset is composed of netlists with unseen architectures to evaluate the generalization ability. Specifically, we use adder/multiplier designs from [25], whose architectures are distinguished from training ones. The test subtractor designs are generated similarly as training data but with different constraints and thus disparate architectures. The netlists operate on word lengths ranging from 8 to 32 bits, with the number of gates ranging from hundreds to thousands. The statistic of the datasets is shown in Table 3. For evaluation, we use accuracy as the performance metric.

We reimplemented several representative prior works [5, 7, 14] as the baseline methods for comparison. These works have covered

Table 3: Statistics of the dataset for netlist classification, including adder, subtractor, multiplier, and divider. We try to avoid involving similar designs 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 4: Summary of performance on netlist classification in terms of accuracy. The second column gives the ratio of the training data size to the testing data size. Our proposed FGNN + NCL framework achieves the best performance on all the cases and suffers from slighter degradation when the training data scale is reduced.

Case	Ratio	GIN [7]	EV-CNN [5]	DVAE [14]	Ours
1	1.3	0.762±0.020	0.904±0.011	0.913±0.005	**0.975±0.008**
2	1	0.745±0.026	0.896±0.009	0.902±0.007	**0.962±0.007**
3	0.7	0.737±0.022	0.884±0.003	0.895±0.009	**0.960±0.009**
4	0.5	0.730±0.015	0.877±0.006	0.885±0.010	**0.951±0.005**
5	0.3	0.725±0.028	0.859±0.015	0.871±0.003	**0.945±0.007**

CNN-based method [5], general GNN-based method [7], as well as customized GNN-based method that adapts to DAGs [14]. All these baselines are trained end-to-end, and we report their performance based on their official implementations.

To thoroughly test the models' generalization ability and robustness against the scarcity of training data, we fix the validation/testing data and train/fine-tune the models with datasets of different scales. The results are summarized in Table 4. From the table, we can see that our proposed framework shows substantial performance superiority over the baseline methods across all the cases. Remarkably, when trained with an unreduced dataset (first row), our framework correctly classifies 97.5% target netlists, achieving a performance gain of 6.2% accuracy over the second-best method [14]. Moreover, our proposed framework suffers from slighter performance degradation when trained with decreasing data size. From the last row of the table, we can see that our method manages to achieve relatively high accuracy (94.5%) even with a training data size that is only 30% of the testing data size, dropped by only 3.1% compared with the best performance. In contrast, the performance of the baseline methods [5, 7, 14] on the same case is decreased by 5.0%, 4.6% and 4.9% respectively.

6 Conclusion

Learning feasible representations from raw gate-level netlists is critical for applying machine learning techniques to EDA. In this work, a novel netlist representation learning framework based on a graph contrastive scheme that extracts basic boolean functionality of netlists is proposed. FGNN, a specialized graph neural network, is further introduced to improve the performance of the framework. Experimental results on in-order and out-of-order RISC-V designs and two distinct downstream tasks verified the framework's effectiveness. The proposed framework can be applied to more downstream netlist tasks, which will be left for future work.

Acknowledgment

This work is partially supported by HiSilicon and The Research Grants Council of Hong Kong SAR CUHK14209420, CUHK14208021.

References

[1] 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.
[2] 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.
[3] 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.
[4] W. Li, A. Gascon, P. Subramanyan, W. Y. Tan, A. Tiwari, S. Malik, N. Shankar, and S. A. Seshia, "Wordrev: Finding word-level structures in a sea of bit-level gates," in *Proc. HOST*. IEEE, 2013, pp. 67–74.
[5] 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.
[6] Z. He, Z. Wang, C. Bai, H. Yang, and B. YU, "Graph learning-based arithmetic block identification," in *Proc. ICCAD*, 2021.
[7] K. Xu, W. Hu, J. Leskovec, and S. Jegelka, "How powerful are graph neural networks?" *Proc. ICLR*, 2018.
[8] W. Hamilton, Z. Ying, and J. Leskovec, "Inductive representation learning on large graphs," in *Proc. NIPS*, 2017, pp. 1024–1034.
[9] P. Veličković, G. Cucurull, A. Casanova, A. Romero, P. Lio, and Y. Bengio, "Graph attention networks," *Proc. ICLR*, 2018.
[10] M. Zhang and Y. Chen, "Link prediction based on graph neural networks," *Proc. NIPS*, vol. 31, pp. 5165–5175, 2018.
[11] 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.
[12] 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.
[13] V. Thost and J. Chen, "Directed acyclic graph neural networks," *arXiv preprint arXiv:2101.07965*, 01 2021.
[14] M. Zhang, S. Jiang, Z. Cui, R. Garnett, and Y. Chen, "D-vae: A variational autoencoder for directed acyclic graphs," *Proc. NIPS*, 2019.
[15] 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.
[16] 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.
[17] 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.
[18] P. Velickovic, 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.
[19] 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.
[20] 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.
[21] Y. Tian, C. Sun, B. Poole, D. Krishnan, C. Schmid, and P. Isola, "What makes for good views for contrastive learning?" *arXiv preprint arXiv:2005.10243*, 2020.
[22] T. Xiao, X. Wang, A. A. Efros, and T. Darrell, "What should not be contrastive in contrastive learning," *arXiv preprint arXiv:2008.05659*, 2020.
[23] 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.
[24] K. Sohn, "Improved deep metric learning with multi-class n-pair loss objective," in *Proc. NIPS*, 2016, pp. 1857–1865.
[25] 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.