Scalable Graph Learning-Based Arithmetic Block Recognition

CUDA
Ziyi Wang, Siting Liu, Yuxuan Zhao, Chen Bai, Yang Bai

Abstract

Learning feasible representation from raw gate-level netlists is essential for incorporating machine learning techniques in logic synthesis, physical design, or verification. We argue that 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 customized graph neural network (GNN) architecture that learns a set of independent aggregators to acquire generic functional knowledge from netlists effectively. Comprehensive experiments on complex real-world designs demonstrate that our proposed solution significantly outperforms state-of-the-art netlist representation learning methods.

1 Introduction

As machine learning (ML) techniques develop rapidly, there is a surge in incorporating ML in electronic design automation (EDA) [1, 2, 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.

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

Preprint. Under review.

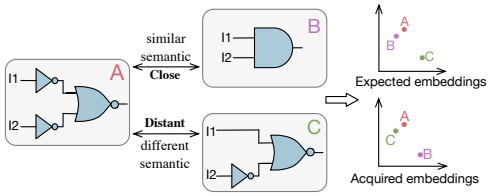

Figure 1: Illustration of the drawback of existing structure-based netlist representation learning methods.

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. 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 propose a customized graph neural network architecture to to extract the basic logic functionality of netlists, which is universal and transferable across different designs. 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:

- 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.

The rest of the paper is organized as follows: 2 introduces some preliminaries. 3 gives the problem formulation. 4 introduces our proposed netlist representation framework. 5 presents experimental evaluations of our proposed framework. 6 concludes the paper.

2 Preliminaries

2.1 Graph Neural Network

Graph neural networks (GNNs) [7, 8, 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

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

where $\vec{h}_v^{(k)}$ 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, 11, 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.

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.*

4 Netlist Representation Learning Framework

4.1 Customized Graph Neural Network: FGNN

As mentioned in 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 crucial to achieving better performance. This part discusses how to design a novel graph neural network architecture that extracts the prior knowledge of netlists' basic logic functionality.

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' 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 2.

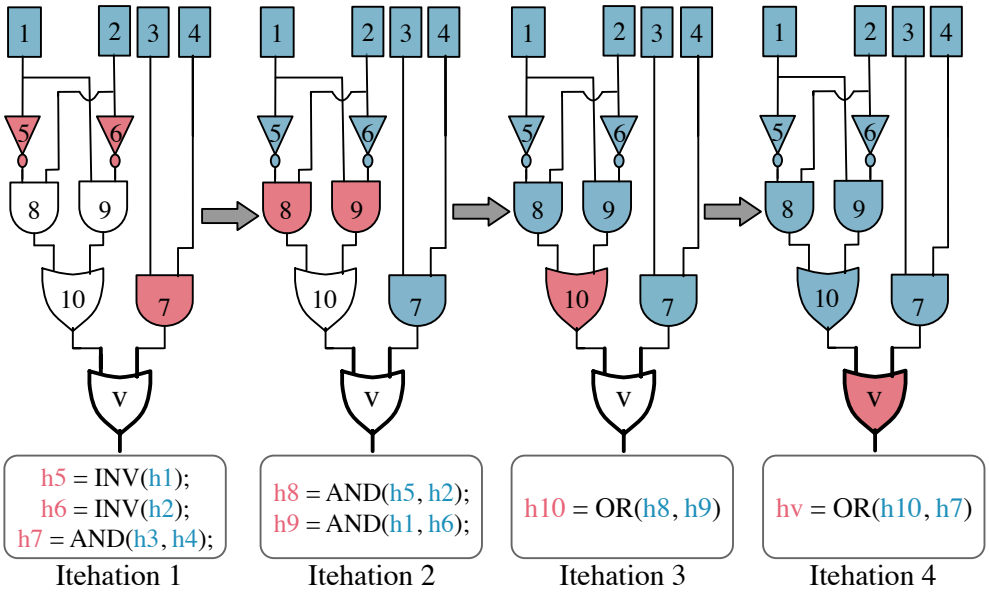

Figure 2: An illustration of the representation generating procedure. The target vertex v is emphasized with bold outlines. Square vertices are primary inputs initializing with all-one representations. Different states of vertices are indicated by fill colors: blank means inactive, red means being computed at current iteration and blue means having been computed in previous iterations. INV, AND, OR are different aggregators.

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

$$\begin{aligned} \vec{h}^{(k)}_{\{\mathbf{i}:\mathcal{D}(\mathbf{i},\mathbf{v})=\delta-\mathbf{k}\}} &= \mathcal{A}_{\{\mathbf{g}:\mathcal{T}(\mathbf{i})=\mathbf{g}\}}(\{\vec{h}_u^{(k-1)} : u \in \mathcal{N}(i)\}) \\ &= \sigma(W_g \cdot f_g(\{\vec{h}_u^{(k-1)} : u \in \mathcal{N}(i)\})), \end{aligned} \tag{2}$$

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 $\vec{h}_{global}$, as described in 3,

$$\vec{h}_{global} = \text{READOUT}(\{\vec{h}_u : u \in V\}) \tag{3}$$

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

5 Experiments

5.1 Setting Up

We implemented the netlist representation learning framework with DGL [5], a graph learning library based on PyTorch. FGNN is trained/tested 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.

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	
		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**
2	2/6	0.612	0.605	0.758	0.757	0.734	0.74	**0.784**	**0.788**
3	3/6	0.633	0.615	0.854	0.865	0.877	0.881	**0.916**	**0.914**
4	4/6	0.662	0.637	0.883	0.889	0.921	0.917	**0.931**	**0.933**
5	5/6	0.738	0.648	0.905	0.898	0.927	0.922	**0.952**	**0.944**
6	6/6	0.768	0.655	0.919	0.917	0.945	0.941	**0.963**	**0.952**

Table 2: Statistics of the dataset for sub-netlist identification with 6 different types of adders. We use *BOOM* as the training set as it is more complicated, leaving *Rocket* as the testing set.

Architecture	*Rocket*		*BOOM*	
	#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

The performance of our proposed framework is evaluated on a specific netlist task, arithmetic block identification. We feed the downstream target netlists into our FGNN to generate node representations, which are then fed into a classifier (Multilayer Perceptron, MLP) to make predictions.

5.2 Evaluation on Arithmetic Block Identification

5.2.1 Background

We 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, where the performance is measured in terms of recall and F1-score.

5.2.2 Dataset

We follow the same experimental setting as [6], where the dataset comes from open-source RISC-V CPU designs [15], including *Rocket* [16], a 5-stage in-order scalar core, and Berkeley Out-of-Order (*BOOM*) Core [16], an out-of-order superscalar RV64G core. Since *BOOM* is more complicated (around 5x larger than *Rocket*), we use it as the training set, while leaving *Rocket* as the testing set.

The netlists are automatically generated from Chisel, which is further synthesized with Synopsys Design Compiler targeting the SAED 32/28nm Digital Standard Cell Library. For each design, we synthesize a set of netlists using various design constraints, so that different adder designs could be generated by DC.

The detail of the dataset we use is shown in 2.

Table 3: Compared with traditional methods

	TETC'13 [18]	DATE'15 [17]	ours
F1-score	0.565	0.435	0.952

5.2.3 Baselines

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.

5.2.4 Results

To evaluate the models' generalization ability, we fix the testing/validation dataset containing all the six different adder architectures (shown in 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 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 [8, 6] 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). This result shows the effectiveness of our proposed framework in extracting high-level prior knowledge of netlists.

We also compare our method with some traditional methods for identifying adders. We choose two representative works: a structural one [17] and a functional one [18]. The results are shown in 3

6 Conclusion

Learning feasible representations from raw gate-level netlists is critical for applying machine learning techniques to EDA. In this work, we propose a specialized graph neural network FGNN to extract the basic functionality of netlists, which is omitted by previous GNN-based works. Experimental results on in-order and out-of-order RISC-V designs verified the framework's effectiveness.

7 Workload Distribution

We assign the workload evenly to each group member as follows:

- Chen Bai: Dataset (generize the netlist designs, transform netlist to DAGs, labeling the adders from the netlists).
- Ziyi Wang: Survey of related works (netlist representation learning), implement our proposed FUNN and baseline method ABGNN, write train/test script.
- Yuxuan Zhao: Survey of related works (GNN), implement baseline methods GraphSAGE, EV-CNN.
- Siting Liu: Survey of background information (problem fomulation), run the experiments, train and test the models (FGNN case 4-6, GraphSAGE).
- Yang Bai: Survey of background information (problem fomulation), run the experiments, train and test the models (FGNN case1-3, ABGNN).

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] 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.

[16] 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.

[17] X. Wei, Y. Diao, T.-K. Lam, and Y.-L. Wu, "A universal macro block mapping scheme for arithmetic circuits," in *Proc. DATE*, 2015, pp. 1629–1634.

[18] P. Subramanyan, N. Tsiskaridze, W. Li, A. Gascón, W. Y. Tan, A. Tiwari, N. Shankar, S. A. Seshia, and S. Malik, "Reverse engineering digital circuits using structural and functional analyses," *IEEE TETC*, vol. 2, no. 1, pp. 63–80, 2013.