DeepGate: Learning Neural Representations of Logic Gates

Min Li*
The Chinese University of Hong Kong

Sadaf Khan*
The Chinese University of Hong Kong

Zhengyuan Shi
The Chinese University of Hong Kong

Naixing Wang
Huawei Technologies Co., Ltd.

Huang Yu
Huawei Technologies Co., Ltd.

Qiang Xu
The Chinese University of Hong Kong

ABSTRACT

Applying deep learning (DL) techniques in the electronic design automation (EDA) field has become a trending topic. Most solutions apply well-developed DL models to solve specific EDA problems. While demonstrating promising results, they require careful model tuning for every problem. The fundamental question on *"How to obtain a general and effective neural representation of circuits?"* has not been answered yet. In this work, we take the first step towards solving this problem. We propose *DeepGate*, a novel representation learning solution that effectively embeds both logic function and structural information of a circuit as vectors on each gate. Specifically, we propose transforming circuits into unified and-inverter graph format for learning and using signal probabilities as the supervision task in DeepGate. We then introduce a novel graph neural network that uses strong inductive biases in practical circuits as learning priors for signal probability prediction. Our experimental results show the efficacy and generalization capability of DeepGate.

KEYWORDS

Representation Learning, Graph Neural Networks, Logic Gates

ACM Reference Format:
Min Li, Sadaf Khan, Zhengyuan Shi, Naixing Wang, Huang Yu, and Qiang Xu. 2022. DeepGate: Learning Neural Representations of Logic Gates. In *Proceedings of the 59th ACM/IEEE Design Automation Conference (DAC) (DAC '22), July 10–14, 2022, San Francisco, CA, USA.* ACM, New York, NY, USA, 6 pages. https://doi.org/10.1145/3489517.3530497

1 INTRODUCTION

The rise of deep learning (DL) has aroused much interest in applying it to solve various electronic design automation (EDA) problems [13]. The most natural representation of circuits and netlists is a graph. With the recent success of graph neural networks (GNNs) [10, 14] in modeling non-structured data, various works have explored its potential on EDA problems such as congestion prediction [15] and testability analysis [16]. These works focus on learning a particular function that takes the circuit graph as input and directly maps it to output for desired EDA tasks, without considering the internal computational process in the circuits.

Recently, a notable trend in the deep learning community is to employ pre-trained models for many downstream tasks rather than learning a specific model for each task from scratch [11]. For example, a series of convolutional neural networks (CNNs) are pre-trained on the ImageNet dataset. They perform well on other computer vision (CV) tasks such as image segmentation and object detection by fine-tuning with a small amount of task-specific data. Similarly, pre-trained Transformer-based language models (e.g., GPT [6] and BERT [8]) have achieved unparalleled performance on various natural language processing (NLP) tasks.

Whereas in the EDA domain, despite all the recent efforts in learning-based solutions [13], obtaining a *general and effective circuit representation* that serves as the basis for solving various EDA tasks has not been addressed yet. In this work, we take the first step towards this direction by introducing a novel GNN-based solution for the representation learning of logic gates, namely *DeepGate*, which is aware of the logic computation procedure and the structural information of combinational circuits.

Naturally, logic circuits can be modeled as directed acyclic graphs (DAGs), in which logic gates appear in a specific topological order. Therefore, one could collect many logic circuits and resort to existing DAG-GNN architecture [21, 27] to learn the node embedding for each logic gate with some supervision tasks (e.g., Boolean satisfiability [3]). However, we argue that such straightforward solutions cannot effectively extract information from circuit graphs.

Firstly, logic circuits could follow different design styles and use diverse technology libraries containing various logic gate types, leading to heterogeneous circuit graphs with mixed distributions that are challenging to learn. In DeepGate, we propose to conduct learning on a general intermediate representation of logic circuits, i.e., *and-inverter graph* (AIG), with the help of logic synthesis tools [5]. The benefits are twofold: (i). Such a unified format constrains the circuit graph distribution without changing circuit functionalities. All the transformed circuits only feature two types of logic gates (i.e., 2-input AND gate and inverter); (ii). The logic synthesis procedure naturally introduces a strong inductive bias of practical circuits for effective learning with GNN models.

Secondly, the effectiveness of representation learning heavily relies on supervision tasks. For example, when pre-training CV models, the image class labels of the ImageNet dataset serve as the cornerstone. In contrast, self-supervised tasks are used instead in NLP pre-trained model development. For effective circuit representation learning, we propose to use the signal probability (i.e., the probability of being logic '1') for every node as rich supervision because it embeds the genuine logic relationship of each node in the circuits. To be specific, we perform logic simulations with a large amount of random patterns to obtain faithful probability values for supervision.

*Both authors contributed equally to this research.

DAC '22, July 10–14, 2022, San Francisco, CA, USA

ACM ISBN 978-1-4503-9142-9/22/07…$15.00
https://doi.org/10.1145/3489517.3530497

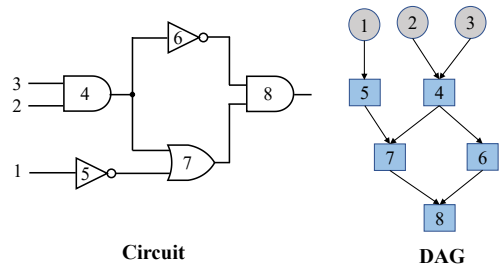

Figure 1: The Circuit Representation as DAG

Last but most important, existing GNN models are general solutions designed to extract information from all kinds of graphs, while circuit graphs are a unique type of graph with logic relationships between nodes. In this work, we design a dedicated GNN model for circuit graphs, significantly enhancing the learning effectiveness. We summarize the contributions of this work as follows:

- To the best of our knowledge, DeepGate is the first work for the *general and effective circuit representation learning* problem. Specially, we propose a novel design flow to tackle this problem: (i). circuit transformation in AIG form; (ii). supervision with logic-simulated probabilities; (iii). representation learning with a dedicated GNN model for circuit graphs.
- We propose a novel GNN model for circuit graphs that exploits unique circuit properties, including *attention mechanisms* that mimic the logic computation procedure and *reversed propagation layers* that consider logic implication effects.
- Reconvergence structures are inevitable due to logic sharing in multi-level logic networks, and they are the main challenges for logic analysis [17]. We treat them as first-class citizen and introduce novel solutions in our GNN model.

We learn the representations of logic gates with many small subcircuits extracted from benchmark circuits. Experimental results performed on large circuits show the efficacy and generalization capability of DeepGate. We organize the remainder of this paper as follows. We review related works in Section 2. Section 3 introduces the DeepGate architecture, while in Section 4, we present the experimental results on various circuits. Finally, Section 5 concludes this paper.

2 RELATED WORKS

2.1 Graph Neural Networks

Graph neural networks [10, 14] have received a lot of attention for their effectiveness in modeling non-structured data. By learning vectorial representations on graphs via feature propagation and aggregation, GNNs show convincing results in various domains [9, 12, 24]. The most popular GNN model employs a message-passing neural network architecture, which computes representation/hidden states $\mathbf{h}_v^\ell$ for node v in a graph $\mathcal{G}$ at every layer ℓ and a final graph representation $\mathbf{h}_\mathcal{G}$, as in [9]:

$$\mathbf{h}_v^\ell = \text{COMBINE}^\ell(\mathbf{h}_v^{\ell-1}, \text{AGGREGATE}^\ell(\{\mathbf{h}_u^{\ell-1} | u \in \mathcal{N}(v)\})), \ell = 1, ., L \quad (1)$$

$$\mathbf{h}_\mathcal{G} = \text{READOUT}(\{\mathbf{h}_v^L, v \in \mathcal{V}\}) \quad (2)$$

wherein $N(v)$ denotes neighboring nodes of node v and L is the number of layers. The parameterized function AGGREGATE^ℓ aggregates messages from neighboring nodes $N(v)$, and COMBINE^ℓ obtains an updated hidden state after aggregation. Finally, the function READOUT^ℓ retrieves the states of all nodes $\mathcal{V}$ and produces the graph neural representation. A notable GNN architecture is the graph attention network (GAT) [23] that considers the importance of different neighbors during aggregation.

Directed acyclic graphs (DAGs) are a special type of graphs, yet broadly seen across many domains, including circuit modeling (see Fig. 1). Recently, few studies have been dedicated to DAG-GNN designs [21, 27], which propagate the message following the topological ordering between nodes and only consider the predecessors in the AGGREGATE^ℓ function, as demonstrated in Equation (3).

$$\mathbf{h}_v^\ell = \text{COMBINE}^\ell(\mathbf{h}_v^{\ell-1}, \text{AGGREGATE}^\ell(\{\mathbf{h}_u^\ell | u \in \mathcal{P}(v)\})), \ell = 1, ..., L \quad (3)$$

The major difference between Eq. (3) and Eq. (1) is that in DAG-GNN, the aggregation function for v will be only executed after all of its predecessors' hidden states have already been computed.

Besides stacking L layers to increase the depth of the network, one can also apply the same model for T times in the *recurrent* fashion to generate the final embedding [3]:

$$\mathbf{h}_v^t = \text{COMBINE}(\mathbf{h}_v^{t-1}, \text{AGGREGATE}(\{\mathbf{h}_u^t | u \in \mathcal{P}(v)\})), t = 1, ..., T \quad (4)$$

Using the taxonomy defined in [25], we name the two variants of DAG-GNNs described in Equations (3)–(4) as *DAG-ConvGNNs* and *DAG-RecGNNs*, respectively.

2.2 GNN-Based Solutions for EDA Problems

Existing GNN-based EDA solutions use an end-to-end flow for specific EDA tasks wherein the labels are usually extracted from commercial EDA tools.

To the best of our knowledge, the first GNN-based EDA technique is applied to the test point insertion (TPI) problem, which is formulated as a node binary classification problem and solved with a graph convolutional network [16]. The ground-truth labels are collected from commercial TPI tools, revealing whether a particular node is "easy to observe" or not. CongestionNet [15] models the circuit as an undirected graph and trains a GAT model to predict the congestion of the final physical design on a per-cell basis. GRANNITE [28] conducts power estimation using a DAG-GNN model. Gate netlists are mapped onto graphs with per-node (gate) and per-edge (net) features. They achieved good accuracy (less than 5.5% error across a diverse set of benchmarks) for fast (<1 second) average power estimation on designs up to 50k gate. Recently, [26] proposes a GAT-based model named Net^2 for pre-placement net length estimation.

To solve a particular EDA problem, the above techniques typically pre-compute many node/edge features (e.g., SCOAP testability measures in [16]) and use existing GNN models to aggregate these features for solution findings. Consequently, the learned node features cannot be transferred among related tasks, despite using the same circuit graphs as inputs. More importantly, an effective representation for circuits should be aware of their logic functions. However, existing solutions ignore it and only consider the structural information in their learning procedure.

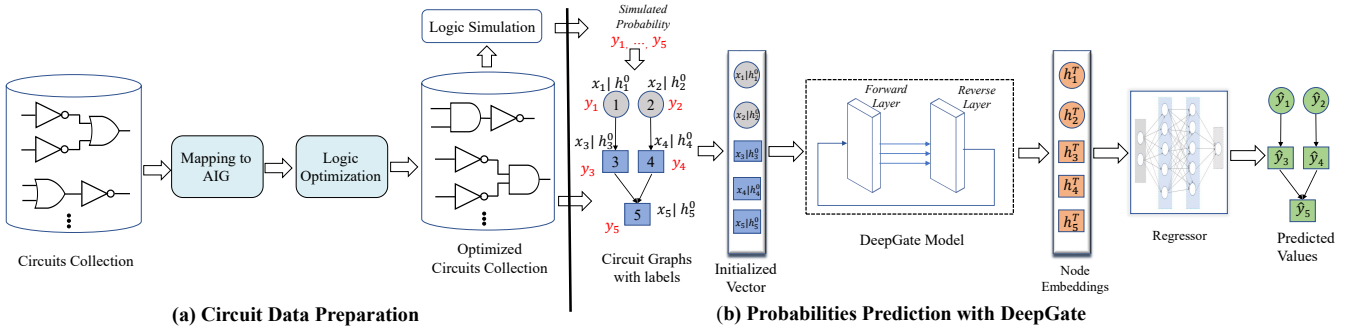

Figure 2: The Overview of DeepGate.

Motivated by the above, we propose to learn a general and effective circuit representation without pre-computing any specific features, as detailed in the following section.

3 PROPOSED SOLUTION

3.1 Overview of DeepGate

Figure 2 presents the overview of the proposed DeepGate solution, consisting of two stages for the neural representation learning of logic gates:

- **Circuit Data Preparation:** Given a pool of circuit designs, we use logic synthesis tools to transform them into a unified AIG format. We then perform logic simulations on the circuits with sufficient random patterns to obtain the signal probability (i.e., the probability of node being logic '1') on every node as supervision. We elaborate the details in Section 3.2.
- **Probabilities Prediction with DeepGate:** Given a circuit dataset and the logic-simulated probabilities as the supervision task, we introduce a novel GNN model dedicated for circuit graph analysis to learn the neural representations of logic gates, as detailed in Section 3.3.

3.2 Circuit Data Preparation

Some circuits are at the register-transfer level, while others are gate-level netlists mapped with various libraries. Such heterogeneity across circuits is a challenge for GNN model development. To tackle this problem, we resort to the logic synthesis tool ABC [5] and transform all circuits into the unified AIG format. If the original circuit is too large, we extract small sub-circuits with circuit sizes ranging from 30 to $3k$ gates. Note that, we test the effectiveness of DeepGate on much larger circuits for its generalization capabilities.

The benefits of such circuit pre-processing flow include: (i). Only two logic gate types (i.e., 2-input *AND* gate and 1-input *Not* gate) are considered, which would dramatically reduce representation learning difficulty; (ii). Applying logic synthesis introduces strong relational inductive bias into the resulting circuit graphs; (iii). The constraint on circuit size facilitates efficient GNN training with both reduced sizes of circuit graphs and less time for preparing supervision labels.

There are many possibilities to annotate a circuit, e.g., the satisfiability of the circuit [3]. However, a good supervision task should satisfy the following condition: the labels should be easily obtained while retaining rich information for both the logic function and the structural information of the circuits. In DeepGate, we propose to use the signal probability on every node as supervision, which satisfies the above requirements: (i). It is relatively easy to obtain highly-accurate probability values by running logic simulations on many random input patterns, especially when the circuit size is limited; (ii). A unique yet important property of logic circuits that makes circuit analysis challenging is the reconvergence structures, and logic simulation is arguably the only way to obtain the actual value for such structures; (iii). The logic probability of each gate itself plays an essential role in many EDA tasks.

3.3 GNN Model in DeepGate

Given circuit graphs in AIG form, the objective of our GNN model is to estimate the probability of every node such that it would be as close to the genuine signal probability as possible. Different from existing DAG-ConvGNNs [21, 27] and DAG-RecGNNs [3] models that focus on learning the topological information in the graph, DeepGate is designed to learn both the circuit structural information and the computational behaviour of logic circuits, and embed them as vectors on every logic gate.

We now elaborate on the detailed GNN model design in DeepGate. Given a circuit graph $\mathcal{G}$, we embed the gate type of each node v with one-hot encoding in $\mathbf{x}_v$. To be specific, as only primary inputs (PIs), *And* gates and *Not* gates are present in AIGs, we assign a 3-d vector for each node according to its gate type. It should be noted that instead of relying on the probability-based measurements in previous works [16, 28], our model only requires gate type information for the representation learning. We also have hidden states $\mathbf{h}_v$ for every node, which is initialized randomly. Given these, DeepGate resorts to attention-based aggregation design [21, 23] and the gated recurrent unit (GRU) [27] as the update function.

Aggregation. We use the attention mechanism in the additive form to instantiate the AGGREGATE function in Equation (3), wherein the aggregated message $\mathbf{m}_v^t$ for a node v at t^{th} iteration is computed by:

$$\mathbf{m}_v^t = \sum_{u \in \mathcal{P}(v)} \alpha_{uv}^t \mathbf{h}_u^t \quad \text{where} \quad \alpha_{uv}^t = \underset{u \in \mathcal{P}(v)}{softmax}(w_1^\top \mathbf{h}_v^{t-1} + w_2^\top \mathbf{h}_u^t) \quad (5)$$

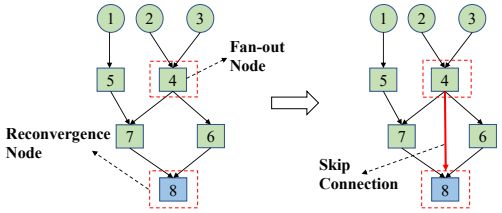

Figure 3: Information Propagation at Reconvergence Node

where α_{uv}^t is a weighting coefficient that is computed by following the query-key design as in usual attention mechanisms. To be specific, $\mathbf{h}_v^{t-1}$ serves as *query*, and representation of predecessors from current iteration t, $\mathbf{h}_u^t$, serves as *key*. The intuition behind using the attention mechanism for aggregation is that when we do the logic computation in digital circuits, the controlling value of a logic gate determines the output of that gate. Therefore, controlling values are far more important than non-controlling values. To mimic this behaviour, the attention mechanism can learn to assign high weights for controlling inputs of gates and give less importance to the rest of the inputs.

Combine. We then use the GRU to instantiate the COMBINE function for updating the hidden state of target node v:

$$\mathbf{h}_v^t = GRU([\mathbf{m}_v^t, \mathbf{x}_v], \mathbf{h}_v^{t-1}) \tag{6}$$

wherein $\mathbf{m}_v^t$, $\mathbf{x}_v$ are concatenated together and treated as input, while $\mathbf{h}_v^{t-1}$ is the past state of GRU.

On the one hand, DeepGate adopts the recursive DAG-GNNs functional defined in Equation (4). The reasons for using the recurrent architecture are two-fold: (i). it is unrealistic for GNNs to capture the circuit's functional and structural information with a single forward propagation; (ii). the recurrent learning procedure facilitates reaching stabilized node embeddings quickly.

On the other hand, our proposed GNN model differs from previous DAG-GNNs [3, 21, 27] that initialize $\mathbf{h}_v^0$ as $\mathbf{x}_v$ and treat the aggregated message as the state of recurrent function. In contrast, we fix the gate type information of nodes $\mathbf{x}_v$ as inputs for all iterations. Such employment can avoid the information vanishing of gate properties during the long-term recursive propagation.

Reversed Propagation Layer. In DeepGate, we also consider backward information propagation, i.e., processing the graph in reversed topological order. One of the main reasons to introduce the backward layers in our framework is that logic implication and backtracking in the reversed order can be highly useful for predicting the states of nodes. It also helps stabilize training, as proved in sequence-to-sequence learning tasks [19].

Regressor. After T iterations, we pass the hidden states of nodes $\mathbf{h}_v^T$ into a multi-layer perceptron (MLP), which computes a single scalar for every node to regress the simulated probabilities. The weights of MLP are shared for nodes with the same gate types. We train the network to minimize the *L1* loss between the prediction $\hat{y}_v$ and the true probability y_v.

3.4 Skip Connection for Reconvergence Structure with Positional Encoding

In the previous section we have described the core components of DeepGate necessary for predicting the logic probabilities of nodes. However, the logic inference on reconvergence nodes is different from normal nodes and such structures are inevitable due to logic sharing in multi-level logic networks. Hence, they are the main challenge for logic probability analysis. To accommodate their impact, we introduce the improvement into DeepGate to enable special processing for reconvergence nodes as shown in Figure 3.

Firstly, we maintain the information of reconvergence nodes during circuit data preparation, including its corresponding source fan-out node, and the logic level difference between the source nodes and reconvergence nodes. Secondly, we add direct edges between the fan-out node and the reconvergence node, named *skip connection* here. The new edges facilitate the information exchange from fan-out nodes to reconvergence nodes. Last but not least, we leverage the positional encoding technique [22] to differentiate the skip connection and the normal connection. To be specific, the function $\gamma(D)$ is a mapping of logic level difference D between source fan-out node and reconvergence node into a higher dimensional space $\mathbb{R}^{2L}$:

$$\gamma(D) = (\sin(2^0\pi D), \cos(2^0\pi D), \cdots, \sin(2^{L-1}\pi D), \cos(2^{L-1}\pi D)) \tag{7}$$

The impact of the fanout node on the reconvergence nodes depends upon the distance between them. Generally speaking, the longer the distance is, the lesser impact it has on the reconvergence node. The above function can induce the knowledge of how much fanout node can impact the result of reconvergence node into the model. We assign the encoded vector as the edge attributes to skip connection and incorporate it into the coefficient calculation described in Equation (5) as the third input.

4 EXPERIMENTS

4.1 Datasets

We extract many sub-circuits from four circuit benchmark suites: ITC'99 [7], IWLS'05 [1], EPFL [2] and OpenCore [20], and follow the circuit data preparation flow described in Section 3.2 to transform all circuits into a unified AIG format. We conduct logic simulations with up to $100k$ random input patterns to obtain an accurate signal probability on every node.

Table 1 presents the statistics of the circuit dataset. #Subcircuits shows the total number of subcircuits extracted from each benchmark. As shown in the table, the constructed circuit dataset covers circuit sizes ranging from tens to thousands of nodes with different logic levels. Finally, there are 10,824 circuits in total, and we create 90/10 training/test splits for model training and evaluation.

4.2 Evaluation Metric and Baselines

To evaluate the performance of different GNN models, we calculate the average value of the absolute differences between the simulated probability y_v and the predicted $\hat{y}_v$ from DeepGate for all nodes $\mathcal{V}$ in the circuits, as shown in equation (8). The smaller the value is,

Table 1: The Statistics of Circuit Training Dataset

Benchmark	#Subcircuits	#Node	#Level
EPFL	828	[52–341]	[4–17]
ITC99	7,560	[36–1,947]	[3–23]
IWLS	1,281	[41–2,268]	[5–24]
Opencores	1,155	[51–3,214]	[4–18]
Total	**10,824**	**[36–3,214]**	**[3–24]**

the better the model performs.

$$Avg.\ Prediction\ Error = \frac{1}{N}\sum_{v\in\mathcal{V}} |y_v - \hat{y}_v| \tag{8}$$

We consider three GNN models: GCN, DAG-ConvGNN, and DAG-RecGNN. *GCN* model treats the circuit graphs as undirected graphs in representation learning. DAG-ConvGNN model follows the settings defined in Equation (3). For DAG-RecGNN model, we adopt the same COMBINE function and the reversed propagation layer design in DeepGate, as depicted in Section 3.3. As for the GNN model in DeepGate, it contains additional attention mechanism and skip connection (SC). Under every setting, we evaluate 4 different aggregator designs, which include representative works for DAG learning, i.e., Convolutional Sum (abbreviated as Conv. Sum) [18], Attention [21, 23], GatedSum [27] and DeepSet [3].

In order to make the comparison fair, we instantiate all models with $d = 64$ for the node hidden states $\mathbf{h}_v$ and design the other parameterized functions to have a similar number of tunable parameters. For DAG-RecGNNs and our DeepGate model, a forward layer is followed by a reversed layer, and $T = 10$ iterations of message passing are performed to obtain the final embeddings. We choose $L = 8$ in Equation (7) for the skip connection setting. For training, all models are optimized for 60 epochs using the ADAM optimizer with a learning rate of 1×10^{-4}. We use the topological batching technique introduced in [21] to accelerate the training.

4.3 Probability Prediction

4.3.1 *Comparison of DeepGate with Baseline Solutions.*

Table 2 compares DeepGate with other baseline solutions in terms of prediction error. From this table, we have several observations: First, both GCN and DAG-ConvGNN are subject to poor performance for probability prediction, mainly due to their lack of ability to model the computational behaviours of circuits. For instance, the best GCN model, equipped with Conv. Sum, gives 0.1386 of prediction error, which in turn is even higher than the worst performing model of DAG-RecGNN. Therefore, only by incorporating the logical ordering into the model design and conducting the propagation recurrently, will make the model perform well. It shows the advantage of DAG-RecGNN implementation with dedicated recurrent scheme and reversed layer design discussed in Section 3.3 over simpler GNN architectures. Second, among all models, DeepGate with attention alone achieves significant prediction error reduction. It brings 22.76% relative improvement compared with the best baseline solution, which is the DAG-RecGNN model equipped with DeepSet aggregator. Hence, using the dedicated attention mechanism benefits logic representation learning. Third, equipped with skip connection design, DeepGate can further reduce the prediction error from 0.0234 to 0.0204, which reveals the efficacy of introducing the reconvergence knowledge into the model design. To summarize, with only the gate type information and the connectivity between gates, DeepGate learns to predict highly-accurate probabilities for logic gates.

Table 2: The Performance Comparison of DeepGate with other GNN models for Logic Probability Prediction

Model	Aggregator	Avg. Prediction Error
GCN	Conv. Sum	0.1386
	Attention	0.1840
	DeepSet	0.2541
	GatedSum	0.1995
DAG-ConvGNN	Conv. Sum	0.2215
	Attention	0.2398
	DeepSet	0.2431
	GatedSum	0.2333
DAG-RecGNN (T=10)	Conv. Sum	0.0328
	DeepSet	0.0302
	GatedSum	0.0329
DeepGate (T=10)	Attention w/o SC	**0.0234**
	Attention w/ SC	**0.0204**

Table 3: The Performance Comparison of DeepGate and DeepSet on Five Large Circuits

Design	#Nodes	Levels	DeepSet	DeepGate	Reduction
Arbiter	23.7K	173	0.0277	0.0073	73.56%
Squarer	36.0K	373	0.0495	0.0346	30.16%
Multiplier	47.3K	521	0.0220	0.0159	27.94%
80386 Processor	13.2K	122	0.0534	0.0387	27.56%
Viper Processor	40.5K	133	0.0520	0.0389	25.18%

As we observe that DAG-RecRNN with DeepSet aggregator (abbreviated as *DeepSet* for the following discussion) performs better than the other baselines, in later results, we only compare DeepGate (w/ skip connection) with it.

4.3.2 *Results on Large Circuits.*

Furthermore, we evaluate DeepGate on five circuit designs that are substantially larger than the circuits it saw during training. The circuit statistics and the prediction error of both DeepGate and DeepSet are shown in Table 3. The number of gates in these designs is two orders of magnitude more than that of the training circuits.

We can observe that DeepGate achieves similar prediction accuracy as that on small circuits, and it outperforms DeepSet in these large circuits by a large margin. Such results clearly demonstrate the generalization capability of DeepGate. In particular, DeepGate achieves 73.56% prediction error reduction on *Arbiter*. This is because, the Arbiter circuit is designed to accommodate access from multiple requests, and it contains repetitive logic units with many reconvergence structures. As DeepGate treats such structures as a first-class citizen in the GNN model, it can generate much more accurate predictions.

Table 4: The Performance of DeepGate with and without Circuit Transformation

	w/o Tran.	w/ Tran.	Pre-trained
EPFL	0.0442	0.0292	0.0142
IWLS	0.0447	0.0342	0.0209

4.4 Discussion

4.4.1 *Effectiveness of Circuit Transformation.*

DeepGate uses the logic synthesis tool to transform the circuits from different sources into unified AIG forms. One may wonder the performance of DeepGate if the network is directly trained on the original circuits, wherein other gate types (e.g., XOR, NAND, NOR, and OR) are also included. To investigate the effectiveness of the circuit transformation in DeepGate, we conduct the controlled experiments on EPFL and IWLS benchmarks, as shown in Table 4.

Take EPFL as an example, we extract 375 sub-circuits from the original designs, and develop two versions: the ones with the original 6 gates types and the other with AIG transformation. For each version, we train the DeepGate model from scratch. The only difference is that for the former version of the dataset, we assign 7-d one-hot encoding for the node feature $\mathbf{x}_v$. As can be observed from Table 4, DeepGate trained on AIGs performs better than the one trained on the original circuits by a large margin (33.94% relative prediction error reduction on EPFL). The same observation can be obtained from the results on IWLS circuits.

Such improvements originate from the benefit of circuit transformation because when only two logic gate types are considered, representation learning difficulty is reduced dramatically without any impact on circuit functionalities. Also, we manually check the usage frequency of different gate types in the original formats, and observe that some gates types (e.g., XOR and NAND) are used much less frequently. Such in-balanced gate distributions may lead to insufficient training, causing higher prediction errors.

Additionally, we directly apply the pre-trained DeepGate model on the merged AIG dataset, as shown in Section 4.1 for comparison. As can be observed, DeepGate trained on the dataset consisting of different benchmarks can further reduce the prediction errors by 51.37%. It supports the claim that unifying different circuit designs into a common intermediate representation can help the model learn a better representation of logic gates.

4.4.2 *Impact of Recurrence Iterations.*

During inference, the number of iterations T can be set as different values. The higher the value, the higher the computational cost. We enumerate T from 1 to 50, and we observe that our GNN model is able to decrease the prediction loss as T increases. However, the prediction error converges quickly at around $T = 10$, despite the circuit size. Such experimental results further demonstrate the scalability of the proposed DeepGate solution.

5 CONCLUSION AND FUTURE WORK

This paper proposes DeepGate, a novel representation learning solution that effectively embeds both logic functions and structural information of a circuit as vectors on each gate. In DeepGate, we construct easy-to-learn circuit graphs by transforming circuits into unified AIG format and introduce a novel GNN model with circuit knowledge as priors for effective representation learning. Using informative signal probability as supervision tasks on small sub-circuits, we show DeepGate can generalize to large circuits with accurate predictions without any pre-computed features.

While showing promising results, the current DeepGate model is still in its infancy. For example, we could introduce other informative supervision tasks (e.g., logic inference and Boolean satisfiability) to achieve better representations for logic gates. We could also add more circuits for training to build a large-scale foundation model for logic circuits [4]. Moreover, in our future work, we plan to apply the representations learned in DeepGate onto many downstream EDA tasks (e.g., power estimation, logic reduction, and equivalence checking). These tasks are directly related to signal probability analysis, and we believe DeepGate can achieve satisfactory results without much effort in finetuning the model.

6 ACKNOWLEDGEMENT

This work was supported in part by Huawei Technologies Co. Ltd.

REFERENCES

[1] Christoph Albrecht. 2005. IWLS 2005 benchmarks. In *IWLS*.
[2] Luca Amarú, Pierre-Emmanuel Gaillardon, and Giovanni De Micheli. 2015. The EPFL combinational benchmark suite. In *IWLS*.
[3] Saeed Amizadeh, Sergiy Matusevych, and Markus Weimer. 2019. Learning To Solve Circuit-SAT: An Unsupervised Differentiable Approach. In *ICLR*.
[4] Rishi Bommasani et al. 2021. On the opportunities and risks of foundation models. *arXiv preprint arXiv:2108.07258* (2021).
[5] Robert Brayton and Alan Mishchenko. 2010. ABC: An academic industrial-strength verification tool. In *CAV*. Springer, 24–40.
[6] Tom B Brown et al. 2020. Language models are few-shot learners. *arXiv preprint arXiv:2005.14165* (2020).
[7] Scott Davidson. 1999. Characteristics of the ITC'99 benchmark circuits. In *ITSW*.
[8] Jacob Devlin et al. 2018. Bert: Pre-training of deep bidirectional transformers for language understanding. *arXiv preprint arXiv:1810.04805* (2018).
[9] J. Gilmer et al. 2017. Neural message passing for quantum chemistry. In *ICML*.
[10] William L Hamilton, Rex Ying, and Jure Leskovec. 2017. Inductive representation learning on large graphs. In *NIPS*.
[11] Xu Han et al. 2021. Pre-trained models: Past, present and future. *AI Open* (2021).
[12] Weihua Hu et al. 2020. Open graph benchmark: Datasets for machine learning on graphs. *arXiv preprint arXiv:2005.00687* (2020).
[13] Guyue Huang et al. 2021. Machine learning for electronic design automation: A survey. *TODAES* (2021).
[14] Thomas N Kipf and Max Welling. 2016. Semi-supervised classification with graph convolutional networks. *arXiv preprint arXiv:1609.02907* (2016).
[15] Robert Kirby et al. 2019. CongestionNet: Routing congestion prediction using deep graph neural networks. In *VLSI-SoC*. IEEE.
[16] Yuzhe Ma et al. 2019. High performance graph convolutional networks with applications in testability analysis. In *DAC*.
[17] MW Roberts and PK Lala. 1987. Algorithm to detect reconvergent fanouts in logic circuits. *IEEE Proceedings Computers and Digital Techniques* (1987).
[18] Daniel Selsam et al. 2018. Learning a SAT Solver from Single-Bit Supervision. In *International Conference on Learning Representations*.
[19] Ilya Sutskever, Oriol Vinyals, and Quoc V. Le. 2014. Sequence to Sequence Learning with Neural Networks. arXiv:1409.3215
[20] Opencores Team. [n. d.]. Opencores. https://opencores.org/.
[21] V. Thost and J. Chen. 2021. Directed Acyclic Graph Neural Networks. In *ICLR*.
[22] Ashish Vaswani et al. 2017. Attention is all you need. In *NIPS*.
[23] Petar Veličković et al. 2017. Graph Attention Networks. *ICLR* (2017).
[24] Le Wu et al. 2018. Socialgcn: An efficient graph convolutional network based model for social recommendation. *arXiv preprint arXiv:1811.02815* (2018).
[25] Zonghan Wu et al. 2020. A comprehensive survey on graph neural networks. *IEEE transactions on neural networks and learning systems* (2020).
[26] Zhiyao Xie et al. 2021. Net2: A Graph Attention Network Method Customized for Pre-Placement Net Length Estimation. In *ASP-DAC*. IEEE.
[27] Muhan Zhang et al. 2019. D-VAE: A Variational Autoencoder for Directed Acyclic Graphs. arXiv:1904.11088
[28] Yanqing Zhang, Haoxing Ren, and Brucek Khailany. 2020. GRANNITE: Graph neural network inference for transferable power estimation. In *DAC*. IEEE.