PolarGate: Breaking the Functionality Representation Bottleneck of And-Inverter Graph Neural Network

Jiawei Liu[1], Jianwang Zhai[1†], Mingyu Zhao[1], Zhe Lin[2], Bei Yu[3], Chuan Shi[1†]
[1]Beijing University of Posts and Telecommunications [2] Sun Yat-sen University
[3]The Chinese University of Hong Kong

Abstract

Understanding the functionality of Boolean networks is crucial for processes such as functional equivalence checking, logic synthesis and malicious logic identification. With the proliferation of deep learning in electronic design automation (EDA), graph neural networks (GNNs) are widely used for embedding the and-inverter graphs (AIGs), a standard form of Boolean networks, into vectorized representation. A key challenge in the use of GNN for Boolean representation is that although GNNs can well encapsulate the structural properties of AIGs, they usually fail to fully capture the functionality of Boolean logic. Moreover, most GNNs designed for AIGs (also called AIGNNs) either rely on a large amount of training data or require complex supervisory tasks, making it difficult to maintain high training efficiency and prediction accuracy. In this work, for the first time, we focus on breaking the bottleneck of AIGNNs by augmenting their capability of functional representation, providing an efficient solution called PolarGate, which naturally aligns the message passing process with the logical functionality of AIGs. Specifically, we map the behavior of the logic gate into an ambipolar state space, customize differentiable logical operators, and design a functionality-aware message passing strategy. Experimental results on two logically related tasks (i.e., signal probability prediction and truth-table distance prediction) show that PolarGate outperforms the state-of-the-art GNN-based methods for Boolean representation, with an improvement of 62.1% (40.6%) in learning capability and 79.5% (85.6%) in efficiency on two tasks. The code is avaliable at https://github.com/BUPT-GAMMA/PolarGate.

1 Introduction

Boolean networks and their simplified forms, And-Inverter Graphs (AIGs), play a pivotal role within the domain of electronic design automation (EDA) [1]. They provide a fundamental representation of logic functions that are essential for the synthesis [2], simulation [3], verification [4], and testing [5] stages. In various tasks related to AIGs, structural modeling and functional modeling are two important aspects. Conventional approaches to processing AIGs encompass structural hashing and functional propagation techniques[6], which frequently encounter scalability issues and struggle to harness modern computational resources [1].

†Corresponding authors.

ICCAD '24, October 27–31, 2024, New York, NY, USA

ACM ISBN 979-8-4007-1077-3/24/10…$15.00
https://doi.org/10.1145/3676536.3676834

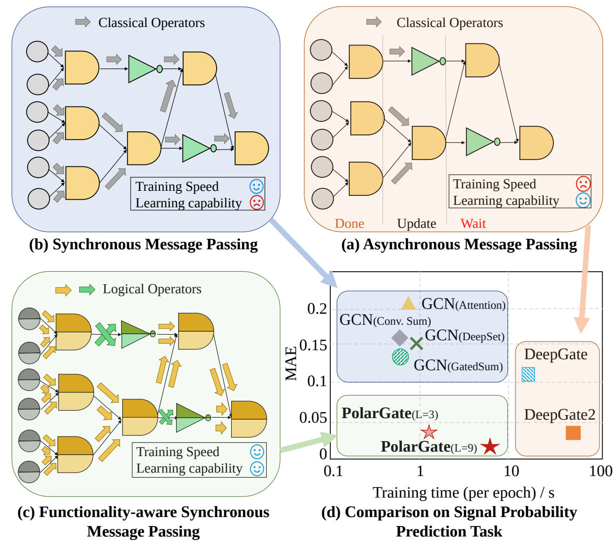


Figure 1: The differences between three message passing paradigms and the comparison between PolarGate and six baseline models.

With the rise of deep learning, using graph neural networks (GNNs) to learn low-dimensional vector representations for each logic gate in AIGs has garnered widespread attention [7]. In this paper, we refer to these GNNs that are tailored for AIGs as And-Inverter Graph Neural Networks (AIGNNs). Specifically, since GNNs can naturally model structural information, early works often design AIGNNs to tasks with strong structural dependencies, such as congestion prediction [8], net length prediction [9] and power prediction [10]. However, a greater number of tasks are closely related to the logical functionality of nodes, such as logic synthesis and formal verification, which require AIGNNs to differentiate the functionality of each logic gate, that is, to possess functional representation ability. To enhance the functional representation ability of AIGNNs, existing methods typically construct complex supervisory information, requiring models to employ multi-task training [1], multi-stage training [2], or additional training data [11]. However, they overlook a key issue: **Is the message passing mechanism currently followed by AIGNNs suitable for learning functionality?**

To answer this question, we conduct an analysis of the message passing mechanism in AIGNNs. The analysis results indicate that there are two inherent limitations in current message passing mechanisms to represent functionality, which significantly restricts the training speed and prediction accuracy of the model. Firstly, since functional modeling often involves long-distance dependencies [12], existing methods [2, 3] tend to deepen the number of GNN layers to enhance the functional representation capability of GNNs. In particular, as shown in Figure 1(a), some methods [2, 3] employ asynchronous message passing, requiring the number of GNN layers to be equal to the network depth. However, deepening the GNN layers can lead to **over-squashing** [13] of information, meaning that information growing exponentially is compressed into a vector of a fixed size. This may result in the loss of significant details and consequently affecting the performance of the model, thereby affecting the model's accuracy. Secondly, the message passing in existing methods does not follow the functional characteristics of logic gates but instead continues to use traditional aggregation operators, such as summation [14] or attention mechanisms [15]. This can result in the representations of AND and NOT gates being too similar, leading to **logical confusion**. We refer to these two issues as the **functionality representation bottleneck** of AIGNNs. It is worth noting that the asynchronous strategy and an overly deep number of layers in AIGNNs also greatly reduce the training efficiency of the model. Although some models [1, 12] use synchronous message passing to improve training and inference efficiency, this is unrelated to enhancing their functionality representation capabilities. On the contrary, as illustrated in Figure 1(b), synchronous message passing may exacerbate logical confusion, leading to poorer predictions [3].

To break the functionality representation bottleneck, we note that in AIGs, each node has ambipolar logical states: high level (1) and low level (0), and the functionality of logic gates is also reflected in the processing of these two states. Considering this prior knowledge has the potential to guide the functional learning of AIGNNs, we propose PolarGate, a novel approach to overcome the functionality representation bottleneck of AIGNNs. As shown in Figure 1(c), PolarGate begins by mapping node features of AIGs into an ambipolar embedding space, thereby distinguishing the underlying two logical states. It then employs custom-designed differentiable logical operators that are aligned with the logical operations inherent to AIGs. Finally, it introduces a functionality-aware message passing strategy that ensures the aggregation process is inherently consistent with the logical functionality of AIGs.

We have conducted extensive experiments on two functionality-related tasks, i.e., signal probability prediction (SPP) and truth-table distance prediction (TTDP), demonstrating that PolarGate outperforms current methods in training efficiency and prediction accuracy. Taking the SPP task as an example, as shown in Figure 1(d), the 9-layer PolarGate has achieved an average reduction of 62.1% in prediction mean absolute error (MAE) compared to the state-of-the-art method, DeepGate2 [2], and has also reduced the training time by 79.5%. In addition, PolarGate can achieve a balance between efficiency and accuracy by adjusting the number of layers.

The main contributions can be summarized as follows:

- Firstly, this paper is the first to study the impact of GNN message passing mechanisms on the functionality representation capabilities of AIGs, and identifies the phenomenon of functionality representation bottleneck.
- Secondly, to break this bottleneck, we propose the ambipolar state principle for the first time. Based on this principle, we design PolarGate, an AIGNN specifically tailored for functional learning, which includes novel embedding spaces, aggregation operators, and message passing strategies.
- Thirdly, we conduct extensive experiments on two functionality-related tasks. Experimental results demonstrate that PolarGate can effectively break the functionality representation bottleneck and significantly improve both training efficiency and prediction accuracy over existing SOTA methods.

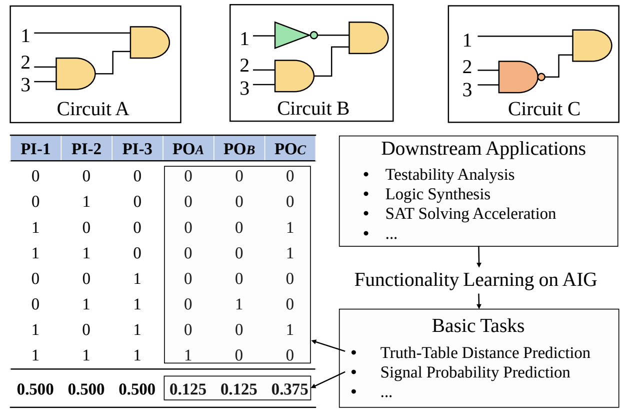


Figure 2: Basic tasks and downstream applications of functionality learning on AIGs.

2 Preliminaries and Related Works

2.1 Boolean Networks and And-inverter Graphs

Boolean Networks refer to networks composed of logical gates such as AND, OR, and NOT, and are widely used in circuit design and verification. Due to the fact that any Boolean network can be systematically decomposed and restructured into AIGs using De Morgan's laws, AIGs are well-suited for various EDA tasks such as rewriting, simulation, and formal verification. The key challenge in processing AIGs is to consider both functional and structural properties simultaneously. Here, structural properties refer to the topological information of the AIG, while functional properties refer to the logical functionality of each gate. As illustrated in Figure 2, these characteristics are not only crucial for basic tasks such as signal probability prediction [3] and truth-table distance prediction [2], but also support more advanced analysis and optimization. Traditional methods of AIG processing include structural hashing and functional propagation [6], which often face challenges in scalability and effective use of modern computing resources, thereby driving the development of AIG representation learning [1–3, 11, 12].

2.2 And-inverter Graph Neural Networks

AIG representation learning currently primarily uses AIGNNs, a type of graph neural networks customized for AIGs. Similar to traditional GNNs, the principle of AIGNNs is to propagate node features along the graph structure and perform transformations in the dimensionality of features during the propagation process, ultimately generating low-dimensional node representations. Since this process simultaneously considers the extraction of structural information and feature aggregation based on the neighborhood, when the node's initial features can strongly encode functional information, it can be regarded as an efficient approximation of structural hashing or functional aggregation [1].

However, in practice, it is not easy to characterize functionality with AIGNNs. This is reflected in the fact that when AIGNNs are applied to functionality-related tasks, existing works still rely on very complex supervisory information to train stably. For instance, Gamora [1] calculates the loss by comparing the AIGNN's output with three carefully designed prediction tasks and is trained via multi-task learning. DeepGate2 [2] employs curriculum learning to allow the AIGNN to first learn simpler tasks before tackling more difficult ones. FGNN [11] generates additional AIG data based on Boolean equivalence and uses contrastive learning for pre-training.

2.3 Message Passing in AIGNNs

Just as introduced in the previous section, existing works invariably emphasize the important role of these training techniques in improving prediction accuracy, but they do not explain why the performance would deteriorate without these training techniques. This inspires us to consider a more fundamental question, namely, **whether existing AIGNNs are suitable for representing functionality without relying on specific training techniques?** To answer this question, we delve into a more in-depth introduction of the underlying mechanisms of AIGNNs, namely the message passing process, where AIGNN primarily includes two paradigms: asynchronous message passing [2, 3] and synchronous message passing [1, 12]. Since an AIG is a directed acyclic graph (DAG), a natural idea is asynchronous message passing, where node representations are updated sequentially following the topological order of the AIG [2, 3].

However, the asynchronous strategy significantly increases the computational complexity and training costs during the training phase. Additionally, asynchronous message passing requires setting the number of layers in the GNN to match the depth of the AIG, which can lead to various issues when the scale of the AIG is large, such as over-smoothing [16] and over-squashing [13]. To avoid these problems, some works [1, 12] have tried using synchronous message passing, which updates all node representations simultaneously based on their neighboring information. This method has high training efficiency and scalability [1, 12], and performs exceptionally well in certain downstream tasks (e.g., addition tree extraction [1] and QoR prediction [12]). However, since functional information can easily get confused during synchronous message passing, for tasks that heavily rely on functional information, these models often underperform compared to asynchronous message passing [3].

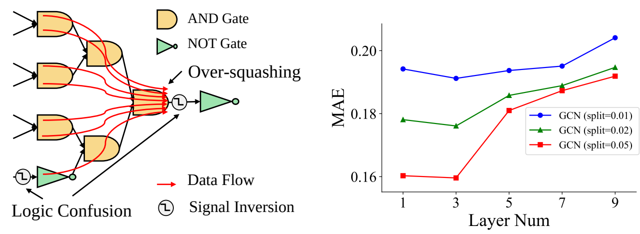
2.4 Problem Formulation

Let $G = (V, E, \mathbf{X})$ be an AIG, where $V = \{v_1, v_2, \ldots, v_n\}$ represents the set of n nodes classified into three types: Primary Inputs (PIs), AND nodes, and NOT (Inverter) nodes. $\mathbf{X} \in \mathbb{R}^{n \times d^{in}}$ is the node feature matrix where each row $X_v \in \mathbb{R}^{d^{in}}$ represents the d^{in}-dimensional features of a node v, which are typically obtained through one-hot encoding of the node type. $E \subset V \times V$ denotes the sets of directed edges, where PIs only act as source nodes for directed edges. We use $\mathbf{A} \in \mathbb{R}^{n \times n}$ to denote the adjacency matrix of the graph G, where $\mathbf{A}_{ij} = 1$ signifies a directed edge from v_i to v_j, and $\mathbf{A}_{ij} = 0$ otherwise (indicating no edge from v_i to v_j).

With the aforementioned definitions, the goal of AIG representation learning is to learn a low-dimensional vector representation for each node. Formally, the representation learning can be described by a transformation function $F : (\mathbf{A}, \mathbf{X}) \to \mathbf{Z}$, where F is an AIGNN learned during training that maps adjacency matrix $\mathbf{A}$ and feature matrix $\mathbf{X}$ to a $n \times d^{out}$-dimensional space $\mathbf{Z} \in \mathbb{R}^{n \times d^{out}}$. After training, the parameters in the AIGNN are fixed, and the AIGNN will be used for prediction tasks on unseen AIGs.

3 Motivations

Before formally introducing our method, let us reconsider the key issues in AIG representation learning and thus derive the design motivation for our approach. Firstly, functional modeling has always been the focus of AIG representation learning, but recent research [2] has shown that learning a good functional representation is not easy. In Section 3.1, we believe that this difficulty is related to the message passing mechanism of AIGNNs, which we refer to as the *functionality representation bottleneck* of AIGNNs. Subsequently, to break this bottleneck, we realize that a promising approach is to map each node to two representation spaces, and we will explain the rationality of such a design from multiple perspectives. We refer to this as the *ambipolar state principle* and will introduce it in Section 3.2.



(a) Illustration of Message Passing. **(b) GCN results on different layers.**

Figure 3: Illustration and experimental observations related to the functionality representation bottleneck.

3.1 Functionality Representation Bottleneck

As illustrated in Figure 3(a), we believe that there are two main reasons currently affecting the functional representation capability of GNNs. Firstly, it is challenging to compress exponentially growing information into a vector of fixed size, which is also known as over-squashing [13]. Specifically, learning functional representations often requires the effective use of long-distance information [12]. When using AIGNNs to learn node representations in AIGs, due to the receptive field increasing sharply with the number of AIGNN layers, it is difficult for AIGNNs to effectively obtain information from distant nodes. This phenomenon is widespread in deep GNNs and is closely related to the graph topology [17]. Additionally, some research [18] has shown that simply increasing the number of GNN layers not only does not alleviate this issue but may even lead to the vanishing gradient problem [19].

Secondly, with classical aggregation operators, nodes with opposite functionalities tend to obtain similar node representations during the aggregation process, leading to logical confusion. Specifically, the vast majority of current work on AIG learning directly uses the same aggregation function to process AND gates and NOT gates, especially aggregation functions [14] that have been proven to have strong expressive power in traditional GNNs. However, these aggregation functions do not consider the functional characteristics of the logic gates, that is, how these gates themselves process signals. This logical confusion can also trigger cascading errors with over-correlation [20] or over-smoothing [16] in message passing, fundamentally affecting the optimization of the GNNs.

To demonstrate the functionality representation bottleneck of AIGNN, we conduct a case study on the signal probability prediction task. Figure 3(b) shows the variation of the mean absolute error (MAE) of the Graph Convolutional Network (GCN) [21] with the change of the number of layers under different label rates. From Figure 3(b), it can be seen that as the number of layers increases, the error of GCN does not decrease but increases, indicating that it has not effectively integrated long-distance information. Additionally, some models [2, 3] use asynchronous message passing to aggregate information and require the number of layers in the GNN to be the same as the network depth. Although asynchronous message passing according to logical hierarchy may help alleviate logical confusion, the overly deep layers exacerbate the over-squashing issue and lead to significantly longer training and inference time for the AIGNN model. Therefore, asynchronous message passing models, at the expense of efficiency, do not completely avoid the functionality representation bottleneck.

3.2 Ambipolar State Principle

To break the over-squashing issue in deep GNNs, a common approach in the deep learning community is to add new edges [22] or remove existing edges [23]. However, for AIG representation learning, these methods can cause the message passing to deviate from the original structure of AIGs, thereby exacerbating logic confusion and failing to address functionality representation bottleneck. Moreover, in pursuit of high efficiency, we desire a solution that is applicable to synchronous message passing, which further increases the demand for the ability to disambiguate logic.

To achieve the aforementioned goals, let us consider the key characteristics of AIGs. Compared to DAGs, the characteristic of AIGs is that each node has only two states: low level (expressed as 0) and high level (expressed as 1), representing the logical values output by the node during logic simulation. The propagation of signals across the graph is manifested in the transitions between these two states. Therefore, we believe a simple and direct approach is to map the node representation into two representation spaces instead of a single representation space, which is called the **ambipolar state principle**. There are several motivations for doing this. The first motivation is that in AIGs, AND and NOT gates have inherently different logical functions, which are reflected in their performing different Boolean operations on input signals, following specific rules. When we assign two representation spaces (also called ambipolar embedding space) to each node, we can further design the message passing operators based on the type of node to conform to the Boolean logic. This implies that we are introducing prior knowledge into the message passing process, which can reduce the risk of useful long-distance information being lost during message passing, thereby avoiding over-squashing. The second motivation is that in synchronous message passing, for efficiency, all nodes are updated simultaneously, meaning nodes must update their states without waiting for the states of preceding nodes to be determined. Therefore, the ambipolar embedding space takes into account every possible state of the node, which allows us to avoid the cascading errors caused by logical confusion while ensuring high efficiency.

4 The Proposed PolarGate

Based on the ambipolar state principle, we propose a novel AIGNN framework for AIG representation learning, called PolarGate. As illustrated in Figure 4, PolarGate consists of three components: ambipolar embedding initialization, differentiable logical operators and functionality-aware message passing. Specifically, considering that nodes in AIGs have two logical states, 0 and 1, PolarGate transforms the initial features of each node into two embeddings in positive and negative spaces, seperately. Then, PolarGate defines differentiable logical operators based on Boolean logic, which are more suitable for probabilistic propagation on AIGs compared to traditional operators. Based on this, we define functionality-aware message passing, which conveys function-related information from different embedding spaces using logical operators, thus avoiding logical confusion while ensuring efficiency. After several layers of message passing, the node representations obtained from PolarGate can be used for a variety of functionality-related tasks.

4.1 Ambipolar Embedding Initialization

From the perspective of Boolean logic, nodes in AIGs only exist in two states: 0 or 1. Before message passing, we first transform the initial one-hot features of the nodes into an ambipolar embedding space so that each node has two initial representations, that is, positive embedding and negative embedding. Specifically, we apply different polarization methods to PIs, AND nodes, and NOT nodes, with illustration in Figure 4(a) and distinctions as follows.

For AND nodes, we simply take the average of the neighbors' initial features and concatenate them with their features to form the positive embedding $\mathbf{h}^{Po(1)}$. As for the negative embedding $\mathbf{h}^{Ne(1)}$, we use a zero vector concatenated with the initial features, indicating that no neighbor information is incorporated.

$$
\begin{aligned}
\mathbf{h}_i^{Po(1)} &= \sigma\left(\mathbf{W}^{Po(1)}\Big[\sum_{j\in\mathcal{N}_i}\frac{\mathbf{h}_j^{(0)}}{|\mathcal{N}_i|},\mathbf{h}_i^{(0)}\Big]\right),\\
\mathbf{h}_i^{Ne(1)} &= \sigma\left(\mathbf{W}^{Ne(1)}\Big[0,\mathbf{h}_i^{(0)}\Big]\right),
\end{aligned}
\tag{1}
$$

where $\mathbf{h}_i^{(0)}$ and $\mathbf{h}_j^{(0)}$ represent the initial features of node i and j, $\mathcal{N}_i$ is the set of neighbors of node i. $\mathbf{W}^{Po(1)}$ and $\mathbf{W}^{Ne(1)} \in \mathbb{R}^{d^{hid}\times 2d^{in}}$ are the linear transformation matrices responsible for the positive and negative information. d^{hid} is the size of hidden embeddings, and $\sigma()$ is a non-linear activation function.

Correspondingly, for NOT nodes, we only use the neighbor information to obtain the negative embedding, and we do not incorporate neighbor information when calculating the positive embedding:

$$
\begin{aligned}
\mathbf{h}_i^{Po(1)} &= \sigma\left(\mathbf{W}^{Po(1)}\Big[0,\mathbf{h}_i^{(0)}\Big]\right),\\
\mathbf{h}_i^{Ne(1)} &= \sigma\left(\mathbf{W}^{Ne(1)}\Big[\sum_{k\in\mathcal{N}_i}\frac{\mathbf{h}_k^{(0)}}{|\mathcal{N}_i|},\mathbf{h}_i^{(0)}\Big]\right).
\end{aligned}
\tag{2}
$$

Finally, for PIs, since they do not have any predecessor nodes, we only use their own information to derive two embeddings.

$$
\begin{aligned}
\mathbf{h}_i^{Po(1)} &= \sigma\left(\mathbf{W}^{Po(1)}\Big[0,\mathbf{h}_i^{(0)}\Big]\right),\\
\mathbf{h}_i^{Ne(1)} &= \sigma\left(\mathbf{W}^{Ne(1)}\Big[0,\mathbf{h}_i^{(0)}\Big]\right).
\end{aligned}
\tag{3}
$$

Note that for all nodes, the same parameter matrices $\mathbf{W}^{Po(1)}, \mathbf{W}^{Ne(1)}$ and activation function $\sigma()$ are used to ensure that the embeddings are mapped into the same space.

4.2 Differentiable Logical Operators

In an AIG, the functional characteristics of the nodes are manifested in the transmission of signals based on Boolean logic. For example, an AND gate performs the Boolean intersection operation, with $\mathrm{OP}_{\mathrm{AND}}(0,1) = \mathrm{OP}_{\mathrm{AND}}(1,0) = \mathrm{OP}_{\mathrm{AND}}(0,0) = 0$ and

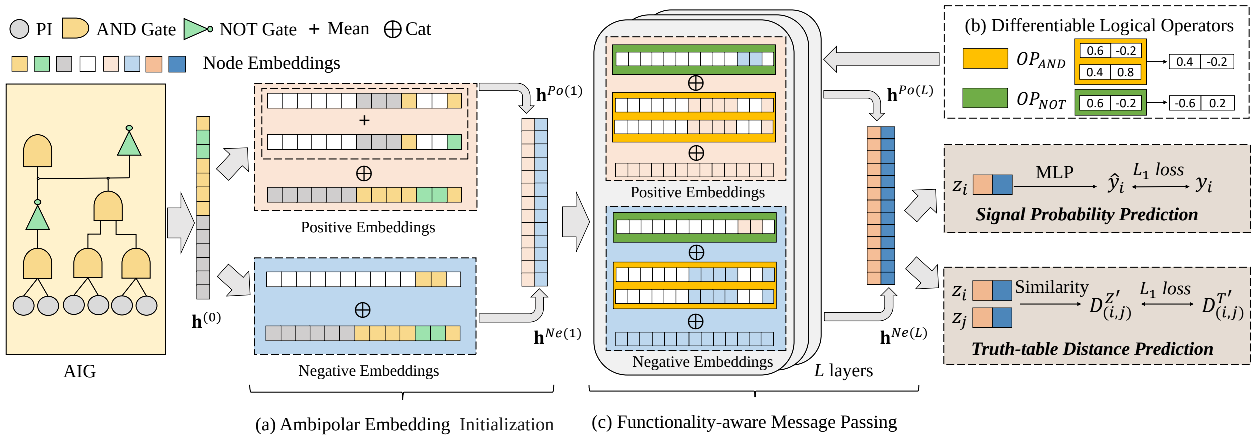


Figure 4: Illustration of PolarGate.

$\mathrm{OP}_{\mathrm{AND}}(1,1) = 1$. A NOT gate performs the Boolean negation operation, with $\mathrm{OP}_{\mathrm{NOT}}(0) = 1$ and $\mathrm{OP}_{\mathrm{NOT}}(1) = 0$. To preserve functional information in AIG representation learning, the message passing in AIGNN should also adhere to Boolean logic. However, since message passing operates in the embedding space that consisting of continuous value vectors, Boolean logic cannot directly process embeddings. To tackle this limitation, we need to devise differentiable logic operators $\mathrm{OP}_{\mathrm{AND}}$ and $\mathrm{OP}_{\mathrm{NOT}}$ for embeddings, and these operators must meet certain conditions:

Proposition 1. *Given the differentiable logical operators* $\mathrm{OP}_{\mathrm{AND}}$ *and* $\mathrm{OP}_{\mathrm{NOT}}$*, node embeddings* $\mathbf{h}$ *should have the following properties:*

- $\mathbf{h}$ is a fixed point of the double application of $\mathrm{OP}_{\mathrm{NOT}}$, that is, $\mathrm{OP}_{\mathrm{NOT}}(\mathrm{OP}_{\mathrm{NOT}}(\mathbf{h})) = \mathbf{h}$.
- Applying the intersection operator $\mathrm{OP}_{\mathrm{AND}}$ repeatedly to multiple instances of $\mathbf{h}$ results in $\mathbf{h}$ itself, i.e., $\mathrm{OP}_{\mathrm{AND}}(\mathbf{h}, \mathbf{h}, \ldots, \mathbf{h}) = \mathbf{h}$.

Many existing GNNs use the bitwise sum as the aggregation operator because this function has been proven to possess expressive capabilities equivalent to the 1-WL test [14]. However, the bitwise sum does not satisfy the two conditions mentioned earlier. To address this, we design the following logic operators: $\mathrm{OP}_{\mathrm{AND}}$ is defined as the bitwise minimum of two embeddings, and $\mathrm{OP}_{\mathrm{NOT}}$ is defined as the bitwise complement of an embedding, as illustrated in Figure 4(b). We will demonstrate that these two logic operators meet the aforementioned conditions:

Proof: For the first property, the negation operator $\mathrm{OP}_{\mathrm{NOT}}$, by its definition of element-wise negation, when applied twice, reverses the effects of the first application, resulting in the original embedding: $\mathrm{OP}_{\mathrm{NOT}}(\mathrm{OP}_{\mathrm{NOT}}(\mathbf{h})) = \mathrm{OP}_{\mathrm{NOT}}(-\mathbf{h}) = -(-\mathbf{h}) = \mathbf{h}$. For the second property, as the intersection operator $\mathrm{OP}_{\mathrm{AND}}$ is defined to take the minimum across the same embeddings, the result trivially remains $\mathbf{h}$, because the minimum of identical elements is the element itself: $\mathrm{OP}_{\mathrm{AND}}(\mathbf{h}, \ldots, \mathbf{h}) = \min(\mathbf{h}, \ldots, \mathbf{h}) = \mathbf{h}$.

4.3 Functionality-aware Message Passing

With the initialized ambipolar embeddings and differentiable logical operators, PolarGate can implement functionality-aware message passing, as illustrated in Figure 4(c). Firstly, when updating the embeddings of AND nodes, PolarGate uses the following aggregation method, which is to use the states of all neighbors with the same polarity and the $\mathrm{OP}_{\mathrm{AND}}$ operator:

$$
\begin{aligned}
\mathbf{h}_i^{Po(l)} &= \sigma\left(\mathbf{W}^{Po(l)}\left[\operatorname*{OP_{AND}}_{j\in\mathcal{N}_i}\mathbf{h}_j^{Po(l-1)}, 0, \mathbf{h}_i^{Po(l-1)}\right]\right),\\
\mathbf{h}_i^{Ne(l)} &= \sigma\left(\mathbf{W}^{Ne(l)}\left[\operatorname*{OP_{AND}}_{j\in\mathcal{N}_i}\mathbf{h}_j^{Ne(l-1)}, 0, \mathbf{h}_i^{Ne(l-1)}\right]\right),
\end{aligned}
\tag{4}
$$

where $\mathbf{W}^{Po(l)}$ and $\mathbf{W}^{Ne(l)}$ are two linear transformation matrices, $\sigma()$ is a non-linear activation function.

For NOT nodes, PolarGate uses the opposite polarity embeddings of its neighboring nodes along with the $\mathrm{OP}_{\mathrm{NOT}}$ operator:

$$
\begin{aligned}
\mathbf{h}_i^{Po(l)} &= \sigma\left(\mathbf{W}^{Po(l)}\left[0, \operatorname*{OP_{NOT}}_{k\in\mathcal{N}_i}\mathbf{h}_k^{Ne(l-1)}, \mathbf{h}_i^{Po(l-1)}\right]\right),\\
\mathbf{h}_i^{Ne(l)} &= \sigma\left(\mathbf{W}^{Ne(l)}\left[0, \operatorname*{OP_{NOT}}_{k\in\mathcal{N}_i}\mathbf{h}_k^{Po(l-1)}, \mathbf{h}_i^{Ne(l-1)}\right]\right).
\end{aligned}
\tag{5}
$$

Note that to distinguish between AND and NOT gates, we use all-zero embeddings at different positions for aggregation, and also consider the node's own state from the previous layer during each aggregation process. For PI nodes, since they do not have predecessors, we adopt the following aggregation method:

$$
\begin{aligned}
\mathbf{h}_i^{Po(l)} &= \sigma\left(\mathbf{W}^{Po(l)}\left[0, 0, \mathbf{h}_i^{Po(l-1)}\right]\right),\\
\mathbf{h}_i^{Ne(l)} &= \sigma\left(\mathbf{W}^{Ne(l)}\left[0, 0, \mathbf{h}_i^{Ne(l-1)}\right]\right).
\end{aligned}
\tag{6}
$$

After the aggregation through L layers, PolarGate concatenates the positive and negative embeddings of each node to form its final representation $\mathbf{Z} = [\mathbf{H}^{Po(L)}, \mathbf{H}^{Ne(L)}]$.

4.4 PolarGate Framework

With the ambipolar embedding space and functionality-aware message passing methods, we can present the entire framework of PolarGate, as shown in Algorithm 1. Firstly, we map the initial features of the AIG nodes into the ambipolar embedding space. Then, based on differentiable logical operators and node type, we define different message passing functions, enabling each node to correctly aggregate polarized information from its neighbors according to its

Algorithm 1 PolarGate Framework

Input: An AIG adjacency matrix $\mathbf{A} \in \mathbb{R}^{n \times n}$; a feature matrix $\mathbf{X} \in \mathbb{R}^{n \times d^{in}}$; number of aggregation layers L.
Output: Low-dimensional representation matrix $\mathbf{Z} \in \mathbb{R}^{n \times d^{out}}$.
Initialize node embeddings $\mathbf{H}^{(0)} \leftarrow (\mathbf{A}, \mathbf{X})$.
Generate ambipolar node embeddings for each type of nodes $(\mathbf{H}^{Po(1)}, \mathbf{H}^{Ne(1)}) \leftarrow (\mathbf{A}, \mathbf{H}^{(0)})$.
while not convergent **do**
  for $l = 1$ **to** $L - 1$ **do**
    Update ambipolar node embeddings for each type of node $(\mathbf{H}^{Po(l)}, \mathbf{H}^{Ne(l)}) \leftarrow (\mathbf{A}, \mathbf{H}^{Po(l-1)}, \mathbf{H}^{Ne(l-1)})$.
  end for
  $\mathbf{Z} \leftarrow [\mathbf{H}^{Po(L)}, \mathbf{H}^{Ne(L)}]$.
  Update PolarGate parameters based on $loss(\mathbf{Z})$ of specific task.
end while

logical function to update its own representation. After L layers of message passing, we concatenate the representations of each node in the ambipolar space to obtain the final node representation, which is used to calculate the loss during the training phase and to make predictions on downstream tasks.

5 Experiments

In this section, we first introduce the experimental setup. Then, we conduct a comprehensive experimental evaluation to sequentially address the following key questions, thereby demonstrating the superiority of PolarGate for the AIG representation learning.

- (Section 5.3 and Section 5.4) Does PolarGate learn functionality-aware node representations that can support multiple tasks?
- (Section 5.5) Does PolarGate perform consistently across different amounts of training data?
- (Section 5.6) Does the PolarGate model's expressive capability improve with increased numbers of layers?
- (Section 5.7) Does each module of PolarGate contribute positively to the final outcomes?

5.1 Evaluation Tasks and Metrics

To verify the capability of functional representation learning, two tasks are chosen to evaluate the performance of different GNN models: **Signal Probability Prediction (SPP)** and **Truth-table Distance Prediction (TTDP)**. SPP is a classic and important task in the field of AIG representation learning [2, 3], while TTDP is a task proposed by DeepGate2 [2] for fine-grained assessment of the functional representation capability of GNNs.

Signal Probability Prediction (SPP). For the SPP task, we compute the average value of the absolute differences between the probability of simulated logic y_v and the predicted $\hat{y}_v$ from AIGNNs for all nodes $\mathcal{V}$ in the circuits, as shown in Equation (7).

$$Avg.\ Error_{SPP} = \frac{1}{N} \sum_{v \in \mathcal{V}} |y_v - \hat{y}_v|. \tag{7}$$

Truth-table Distance Prediction (TTDP). For the TTDP task, two embedding vectors will be similar if the corresponding two node functions are similar. Formally, for node i and node j, we denote the truth table vectors as T_i and T_j, the embedding vectors as z_i and z_j, which can further be expressed as:

$$distance(z_i, z_j) \propto distance(T_i, T_j). \tag{8}$$

Table 1: The Statistics of AIG Datasets [3]

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]**

Specifically, we sample node pairs $\mathcal{N}$ and obtain the Hamming distance of truth table D^T of each node pair:

$$D^T_{(i,j)} = \frac{HammingDistance(T_i, T_j)}{length(T_i)}, (i, j) \in \mathcal{N}. \tag{9}$$

According to Equation (8), the distance of embedding vectors D^Z should be proportional to the Hamming distance of the truth table D^T. Then, we calculate the distance of embedding vectors in Equation (10) based on cosine similarity, since the similarity of embedding vectors should be negatively related to distance $D^T_{(i,j)}$.

$$D^Z_{(i,j)} = 1 - CosineSimilarity(z_i, z_j). \tag{10}$$

The difference between D^Z and D^T can be formulated by:

$$\begin{aligned} D^{T'}_{(i,j)} &= ZeroNorm(D^T_{(i,j)}), \\ D^{Z'}_{(i,j)} &= ZeroNorm(D^Z_{(i,j)}), \\ Avg.\ Error_{TTDP} &= \frac{1}{N} \sum_{(i,j) \in \mathcal{N}} |D^{T'}_{(i,j)} - D^{Z'}_{(i,j)}|. \end{aligned} \tag{11}$$

Two metrics are chosen to facilitate the comparison of different models: *mean absolute error (MAE)* and *average runtime (Avg. RT)*. *MAE* is defined as Equation (7) and Equation (11) for SPP and TTDP tasks, respectively. *Avg. RT* is defined as the average training time per epoch. These metrics can evaluate how well and how fast each model performs in different tasks. Equation (7) and Equation (11) also correspond to the L_1 loss used to guide the training of PolarGate.

5.2 Datasets and Baselines

We utilize the dataset processed in DeepGate [3] and DeepGate2 [2], which is extracted from four circuit benchmark suites: ITC'99 [24], IWLS'05 [25], EPFL [26] and OpenCore [27], and is transformed into a unified AIG format. The statistics of the original dataset are presented in Table 1. The column labeled "#Subcircuits" displays the total number of subcircuits that have been extracted from each benchmark. As indicated in Table 1, the circuit dataset has a wide range of circuit sizes, varying from tens to thousands of nodes with different logic levels. Note that after consulting with the authors of DeepGate2, we learned that some graphs in the original dataset could not be parsed, leaving us with only 9,935 graphs. The default ratio of training/validation/test is set to 5/5/90, because in real-world scenarios, we often encounter a large amount of data that has not appeared in the training set, which puts a greater emphasis on the model's generalization ability.

For comparison, we consider several competitive AIGNN baselines: GCN [21], DeepGate [3] and DeepGate2 [2]. For the GCN model, we evaluate four different aggregator designs, which include representative works for DAG learning, i.e., Convolutional Sum (abbreviated as Conv. Sum) [28], Attention [29, 30], GatedSum [31]

Table 2: Experimental Results on SPP Task

Model	Aggregator	MAE ↓	Avg. RT (s) ↓
GCN	Conv. Sum	0.1596	0.8781
	Attention	0.2030	0.9102
	DeepSet	0.1558	0.9066
	GatedSum	0.1425	0.8197
DeepGate	Attention	0.1105	35.4449
DeepGate2 (Stage 1)	Attention	0.0265	45.3427
DeepGate2 (Stage 1+2)	Attention	0.0256	45.6405
PolarGate (L=3)	Logical	0.0437	3.2355
PolarGate (L=9)	Logical	0.0097	9.3428

Table 3: Experimental Results on TTDP Task

Model	Aggregator	MAE ↓	Avg. RT (s) ↓
GCN	Conv. Sum	0.7192	1.0248
	Attention	0.6237	0.9958
	DeepSet	0.6358	0.8684
	GatedSum	0.6050	0.8617
DeepGate	Attention	0.4159	36.0939
DeepGate2 (Stage 1)	Attention	0.8609	45.3427
DeepGate2 (Stage 1+2)	Attention	0.3825	45.6405
PolarGate (L=3)	Logical	0.2513	3.2038
PolarGate (L=9)	Logical	0.2272	6.5913

and DeepSet [32]. For the DeepGate model, we utilize the full version equipped with an additional attention mechanism and skip connection. DeepGate2 uses two-stage optimization and different loss weights are applied during each optimization stage. In stage 1, DeepGate2 uses two prediction tasks, that is, SPP and reconvergence pair prediction. In stage 2, DeepGate2 utilizes three tasks, including the two tasks from the first stage, as well as the TTDP task. In our experiments, we conduct separate evaluations for each optimization stage: DeepGate2 (Stage 1) represents the model's performance after the initial optimization phase. In contrast, DeepGate2 (Stage 1+2) reflects its performance after completing both optimization stages.

For baselines, we maintain consistency with the hyperparameters specified in their original papers. The node hidden dimension is set to 128 across all models. In DeepGate, the skip connection setting is configured with $L = 8$, and the round is set to 1. Apart from DeepGate2, which we train using the multi-task training approach described in their paper, for PolarGate and all other baselines, we conduct single-task training. All models are trained for a maximum of 500 epochs on a single NVIDIA A800 GPU, with an early stopping strategy using a patience of 50 epochs.

5.3 Signal Probability Prediction

Table 2 shows the experimental results of all models on the SSP task, with PolarGate showing the outcomes for both 3-layer and 9-layer settings. Based on the results, we have the following observations. Firstly, PolarGate demonstrates the best performance with the lowest MAE value of 0.0097, achieving a 62.1% reduction compared to the SOTA baseline. Following closely is DeepGate2 (Stage 1+2) with an MAE of 0.0256, while DeepGate2 (Stage 1) shows a slightly higher MAE at 0.0265. Additionally, we observed that none of the aggregation methods used on GCN achieved the desired results. This is because they have not addressed the functionality representation bottleneck of AIGNNs. Similarly, both DeepGate and DeepGate2 directly employ the Attention aggregation method without specific aggregators tailored to the task. In comparison, PolarGate utilizes functionality-aware message passing and, without the need for complex training methodologies, can achieve better results.

As for runtime, the runtime of DeepGate and DeepGate2 is significantly longer than that of other models, which is due to their use of asynchronous message passing with a deeper model layer and the need for continuous waiting. In contrast, PolarGate has noticeably improved training speed, with a runtime that lies between that of synchronous and asynchronous message passing. Additionally, it can be flexibly adjusted according to the error requirements of the test set, achieving a balance between accuracy and runtime.

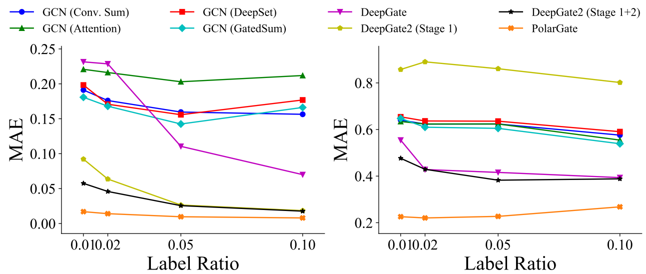
5.4 Truth-table Distance Prediction

Table 3 gives the results of all models on the TTDP task, and we have the following observations. Firstly, PolarGate achieved the best performance in terms of the MAE metric, reducing it by 40.6% compared to the best baseline model, DeepGate2, and also outperforming various variants of GCN. Secondly, the runtime of PolarGate is much lower than that of DeepGate and DeepGate2, and the MAE of the three-layer PolarGate is slightly poorer than that of the nine-layer version. Due to the use of synchronous message passing, PolarGate can appropriately adjust the number of layers according to different tasks, once again demonstrating its advantage in flexibility. Lastly, the performance of DeepGate2 improved after the second stage of training. This is because the second phase of DeepGate2 added optimization for TTDP. It is worth mentioning that compared to the SPP task, the gap between GCN and other models on the TTDP task is not significant, and it even outperforms the DeepGate2 model that was only trained in the first stage. This is because the first stage of DeepGate2 optimizes for SPP and reconvergence structure without considering the TTDP task. This once again illustrates the different optimization directions of SPP and TTDP. Although both are related to logical functionality, a DeepGate2 model trained on SPP cannot be directly applied to the TTDP task.

5.5 Impact of Label Ratio

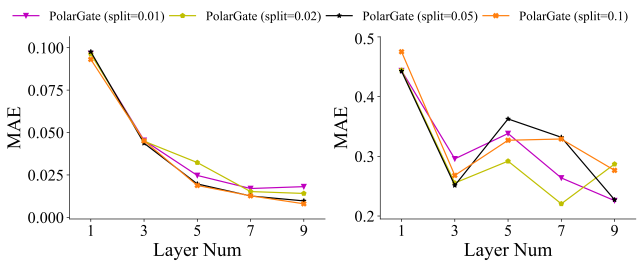
In real-world applications, it is difficult to obtain labeled data in the field of EDA, hence the performance of models at low label rates is particularly important. Additionally, as the amount of available labeled data increases (i.e., the label ratio increases), the performance of the model improves because the model has more information to learn the characteristics of the task. Therefore, a model that performs well at lower label ratios is indicative of higher efficiency in utilizing training data and stronger generalization capabilities. To observe the generalization ability of various models, we set four different label ratios, which are 0.01, 0.02, 0.05 (default), and 0.10. We use an equal scale of data for the validation set, with the remaining data serving as the test set. For instance, a label ratio of 0.02 means we have 2% for the training set, 2% for the validation set, and 96% for the test set. For both the SPP and TTDP tasks, we conduct label ratio experiments and present the results in Figure 5, from which we make the following observations.

Firstly, PolarGate achieves the best performance across all label ratio settings for both tasks. Notably, for the SPP task, the advantage of PolarGate become more pronounced as the label ratio decreased. At a label ratio of 0.01, PolarGate reduces the MAE by 70.4% compared to the best baseline model (i.e., DeepGate2). Secondly, on the SPP task,



(a) SPP (b) TTDP

Figure 5: Experimental results on different label ratios.



(a) SPP (b) TTDP

Figure 6: Experimental results on different GNN layers.

the performance of DeepGate and DeepGate2 improved significantly with the increase in label ratio, while the performance of the GCN models remained almost unchanged. This suggests that although the DeepGate models have an information bottleneck, their performance can be improved with an increase in training data. In contrast, the GCN models, which suffer from both information bottleneck and logical confusion issues, do not see improvements even with an increase in label ratio. This indirectly indicates that PolarGate addresses both the information bottleneck and logical confusion, thereby breaking the functionality representation bottleneck of GNNs overall. Lastly, on the TTDP task, apart from a slight improvement observed in DeepGate and DeepGate2 models when the label ratio increased from 0.01 to 0.02, the performance of the models did not vary significantly across different label ratios. Moreover, the MAEs of all models are consistently and significantly higher in the TTDP task compared to the SPP task. This indicates that TTDP is a more challenging task, and even PolarGate finds it difficult to achieve high performance.

5.6 Impact of Aggregation Layers

Typically, increasing the number of layers in a neural network can enhance the model's learning capability. However, as introduced in Section 3.1, for AIGNNs, issues such as over-squashing and logical confusion can lead to the functionality representation bottleneck when the number of layers is deepened. To explore whether PolarGate can alleviate the functionality representation bottleneck, we conduct experiments with varying numbers of aggregation layers.

The experimental results are presented in Figure 6, and based on these results, we have made the following observations. Firstly, regardless of the label ratio, the MAE of PolarGate on the SPP task decreases as the number of neural network layers increases. It is noteworthy that, as discussed in Section 3.1, due to the functionality representation bottleneck, the MAE of GCN on the SPP task instead increases with the number of neural network layers. This contrast illustrates that PolarGate significantly breaks the functionality representation bottleneck. Secondly, on the TTDP task, apart from a local optimum observed at 3 layers, the MAE also decreases with an increasing number of layers, and the downward trend remains apparent up to 9 layers. The reason why three layers perform well may be related to the evaluation method of the TTDP task, which is based on the relative distance of the embeddings. Therefore, even if the GNN has not learned the absolute precise functionality for each node, the relative distances may be small. In other words, TTDP task [2] still has its limitation when measuring the functionality representation capability of AIGNNs.

Table 4: Ablation Results (MAE) on SPP and TTDP Tasks

Model	SPP	TTDP
PolarGate (w/o Ambipolar Embedding Space)	0.0134	0.2782
PolarGate (w/o Logical Operators)	0.0131	0.3357
PolarGate	**0.0097**	**0.2272**

5.7 Ablation Study

To verify the effectiveness of each module in PolarGate, we present the results of the ablation study in Table 4. Here, as former experiments in Section 5.3 and Section 5.4, we use the same label ratio (i.e., 0.05) and a 9-layer model settings for PolarGate. Due to the inseparable nature of functionality-aware message passing from logical operators, we test two model variants here. PolarGate (w/o Ambipolar Embedding Space) indicates a model that utilizes logical operators and functionality-aware message passing within a single embedding space. In contrast, PolarGate (w/o Logical Aggregators) employs a sum operator and traditional synchronous message passing within an ambipolar embedding space.

Through the experimental results, we can make the following observations: Firstly, for both the SPP and TTDP tasks, the complete model achieved the best results, indicating that each module is useful. Secondly, for the SPP task, the performance difference between the two variants is not significant. However, for the TTDP task, PolarGate (w/o Logical Aggregators) performs worse, which suggests that logical operators are more critical for TTDP.

6 Conclusion

In this work, we explore the functionality representation capability of existing AIGNNs and find that current methods suffer from over-squashing and logical confusion, which we collectively refer to as the functionality representation bottleneck of AIGNNs. To address this issue, we propose the ambipolar state principle and design a novel functionality-aware AIGNN, called PolarGate. PolarGate is designed with ambipolar embedding spaces, differentiable logical operators and functionality-aware message passing strategies. Extensive experiments have demonstrated the effectiveness, efficiency, and generalization of PolarGate. In the future, we hope to apply PolarGate to more AIG-related tasks to demonstrate its superiority.

Acknowledgment

This work is supported by the National Key R&D Program of China (2022YFB2901100), the National Natural Science Foundation of China (No. U20B2045, U1936220, 62192784, 62172052, 62002029, 61772082), the Beijing Natural Science Foundation (No. 4244107), the Research Grants Council of Hong Kong SAR (No. CUHK14210723, CUHK14211824) and the Guangdong Basic and Applied Basic Research Foundation (2023A1515110769).

References

[1] N. Wu, Y. Li, C. Hao, S. Dai, C. Yu, and Y. Xie, "Gamora: Graph learning based symbolic reasoning for large-scale boolean networks," in *ACM/IEEE Design Automation Conference (DAC)*. IEEE, 2023, pp. 1–6.

[2] Z. Shi, H. Pan, S. Khan, M. Li, Y. Liu, J. Huang, H.-L. Zhen, M. Yuan, Z. Chu, and Q. Xu, "DeepGate2: Functionality-aware circuit representation learning," in *IEEE/ACM International Conference on Computer Aided Design (ICCAD)*. IEEE, 2023, pp. 1–9.

[3] M. Li, S. Khan, Z. Shi, N. Wang, H. Yu, and Q. Xu, "DeepGate: Learning neural representations of logic gates," in *ACM/IEEE Design Automation Conference (DAC)*, 2022, pp. 667–672.

[4] H.-T. Zhang, J.-H. R. Jiang, and A. Mishchenko, "A circuit-based sat solver for logic synthesis," in *IEEE/ACM International Conference On Computer Aided Design (ICCAD)*. IEEE, 2021, pp. 1–6.

[5] 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 *ACM/IEEE Design Automation Conference (DAC)*, 2019, pp. 1–6.

[6] 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 Transactions on Emerging Topics in Computing*, vol. 2, no. 1, pp. 63–80, 2013.

[7] 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 Transactions on Design Automation of Electronic Systems (TODAES)*, vol. 26, no. 5, pp. 1–46, 2021.

[8] R. Kirby, S. Godil, R. Roy, and B. Catanzaro, "CongestionNet: Routing congestion prediction using deep graph neural networks," in *IFIP/IEEE 27th International Conference on Very Large Scale Integration (VLSI-SoC)*. IEEE, 2019, pp. 217–222.

[9] Z. Xie, R. Liang, X. Xu, J. Hu, Y. Duan, and Y. Chen, "Net2: A graph attention network method customized for pre-placement net length estimation," in *Asia and South Pacific Design Automation Conference (ASP-DAC)*, 2021, pp. 671–677.

[10] Y. Zhang, H. Ren, and B. Khailany, "GRANNITE: Graph neural network inference for transferable power estimation," in *ACM/IEEE Design Automation Conference (DAC)*. IEEE, 2020, pp. 1–6.

[11] Z. Wang, C. Bai, Z. He, G. Zhang, Q. Xu, T.-Y. Ho, B. Yu, and Y. Huang, "Functionality matters in netlist representation learning," in *ACM/IEEE Design Automation Conference (DAC)*, 2022, pp. 61–66.

[12] C. Deng, Z. Yue, C. Yu, G. Sarar, R. Carey, R. Jain, and Z. Zhang, "Less is more: Hop-wise graph attention for scalable and generalizable learning on circuits," in *DAC*, 2024.

[13] U. Alon and E. Yahav, "On the bottleneck of graph neural networks and its practical implications," in *International Conference on Learning Representations (ICLR)*, 2020.

[14] K. Xu, W. Hu, J. Leskovec, and S. Jegelka, "How powerful are graph neural networks?" in *International Conference on Learning Representations (ICLR)*, 2018.

[15] P. Veličković, G. Cucurull, A. Casanova, A. Romero, P. Liò, and Y. Bengio, "Graph attention networks," in *International Conference on Learning Representations (ICLR)*, 2018.

[16] Q. Li, Z. Han, and X.-M. Wu, "Deeper insights into graph convolutional networks for semi-supervised learning," in *AAAI Conference on Artificial Intelligence (AAAI)*, vol. 32, no. 1, 2018.

[17] J. Topping, F. Di Giovanni, B. P. Chamberlain, X. Dong, and M. M. Bronstein, "Understanding over-squashing and bottlenecks on graphs via curvature," in *International Conference on Learning Representations (ICLR)*, 2021.

[18] F. Di Giovanni, L. Giusti, F. Barbero, G. Luise, P. Lio, and M. M. Bronstein, "On over-squashing in message passing neural networks: The impact of width, depth, and topology," in *International Conference on Machine Learning (ICML)*. PMLR, 2023, pp. 7865–7885.

[19] G. Li, M. Muller, A. Thabet, and B. Ghanem, "DeepGCNs: Can GCNs go as deep as CNNs?" in *IEEE/CVF International Conference on Computer Vision (ICCV)*, 2019, pp. 9267–9276.

[20] W. Jin, X. Liu, Y. Ma, C. Aggarwal, and J. Tang, "Feature overcorrelation in deep graph neural networks: A new perspective," in *ACM SIGKDD International Conference on Knowledge Discovery and Data Mining (KDD)*, 2022.

[21] T. N. Kipf and M. Welling, "Semi-supervised classification with graph convolutional networks," in *International Conference on Learning Representations (ICLR)*, 2016.

[22] M. Black, Z. Wan, A. Nayyeri, and Y. Wang, "Understanding oversquashing in GNNs through the lens of effective resistance," in *International Conference on Machine Learning (ICML)*. PMLR, 2023, pp. 2528–2547.

[23] Y. Liu, C. Zhou, S. Pan, J. Wu, Z. Li, H. Chen, and P. Zhang, "CurvDrop: A ricci curvature based approach to prevent graph neural networks from over-smoothing and over-squashing," in *ACM Web Conference*, 2023, pp. 221–230.

[24] S. Davidson, "Characteristics of the ITC'99 benchmark circuits," in *ITSW*, 1999.

[25] C. Albrecht, "IWLS 2005 benchmarks," in *IWLS*, 2005.

[26] L. Amarú, P.-E. Gaillardon, and G. De Micheli, "The EPFL combinational benchmark suite," in *IWLS*, no. CONF, 2015.

[27] O. Team, "OpenCores," https://opencores.org/.

[28] D. Selsam, M. Lamm, B. Bünz, P. Liang, D. L. Dill, and L. De Moura, "Learning a sat solver from single-bit supervision," in *7th International Conference on Learning Representations, ICLR*, 2019.

[29] P. Veličković, G. Cucurull, A. Casanova, A. Romero, P. Liò, and Y. Bengio, "Graph attention networks," *ICLR*, 2017.

[30] V. Thost and J. Chen, "Directed acyclic graph neural networks," in *International Conference on Learning Representations (ICLR)*, 2021.

[31] M. Zhang, S. Jiang, Z. Cui, R. Garnett, and Y. Chen, "D-vae: a variational autoencoder for directed acyclic graphs," in *Proceedings of the 33rd International Conference on Neural Information Processing Systems*, 2019, pp. 1588–1600.

[32] S. Amizadeh, S. Matusevych, and M. Weimer, "Learning To Solve Circuit-SAT: An Unsupervised Differentiable Approach," in *International Conference on Learning Representations (ICLR)*, 2019.