BoolSkeleton: Boolean Network Skeletonization via Homogeneous Pattern Reduction

Liwei Ni *Student Member, IEEE,* Jiaxi Zhang, Shenggen Zheng, Junfeng Liu, Xingyu Meng, Biwei Xie, Xingquan Li, and Huawei Li *Senior Member, IEEE*

***Abstract*—Boolean equivalence allows Boolean networks with identical functionality to exhibit diverse graph structures. This gives more room for exploration in logic optimization, while also posing a challenge for tasks involving consistency between Boolean networks. To tackle this challenge, we introduce *BoolSkeleton*, a novel Boolean network skeletonization method that improves the consistency and reliability of design-specific evaluations. *BoolSkeleton* comprises two key steps: preprocessing and reduction. In preprocessing, the Boolean network is transformed into a defined Boolean dependency graph, where nodes are assigned the functionality-related status. Next, the homogeneous and heterogeneous patterns are defined for the node-level pattern reduction step. Heterogeneous patterns are preserved to maintain critical functionality-related dependencies, while homogeneous patterns can be reduced. Parameter K of the pattern further constrains the fanin size of these patterns, enabling fine-tuned control over the granularity of graph reduction. To validate *BoolSkeleton*'s effectiveness, we conducted four analysis/downstream tasks around the Boolean network: compression analysis, classification, critical path analysis, and timing prediction, demonstrating its robustness across diverse scenarios. Furthermore, it improves above 55% in the average accuracy compared to the original Boolean network for the timing prediction task. These experiments underscore the potential of *BoolSkeleton* to enhance design consistency in logic synthesis.**

***Index Terms*—Boolean network, skeleton, Boolean dependency, pattern reduction, logic synthesis**

I. Introduction

BOOLEAN networks [1] serve as the intermediate representation in the logic synthesis process [2] within Electronic Design Automation (EDA), where they can be modeled as a typical computational graph. For any given Boolean network, it comprises two key components: "static" functionality and "dynamic" Directed Acyclic Graph (DAG) structure. Here, "static" denotes the functionality that remains invariant for a given design, whereas "dynamic" reflects the variability of the local DAG structure. This variability arises due to the Boolean equivalence theorem [3], which posits that Boolean networks with identical functionality can produce diverse DAG structures as a result of logic optimization techniques. Logic optimization [2] operators aim to reduce the Boolean network's size and depth by the local equivalent replacement techniques [4], [5], thereby enhancing the efficiency of subsequent EDA steps. However, this "dynamic" flexibility poses challenges for functionality-related graph embedding learning in logic synthesis, such as classification [6] and Boolean matching [7]. The variability introduced by optimization complicates the maintenance of consistent representations, creating a tension between structural variance and functional consistency within Boolean networks.

Graph Neural Networks (GNNs) offer a robust framework for learning graph embeddings, effectively extracting consistent features from Boolean networks [8]. Several studies have harnessed GNNs for graph embedding tasks in this domain, including DeepGate and its variants [9], [10], [11], HOGA [12], PolarGate [13], BoolGebra [14], etc. These approaches, however, primarily depend on fine-grained topological structures to represent Boolean networks, placing considerable demands on the expressive capacity of GNNs to capture coarse-grained features. While traditional graph coarsening techniques, such as Variations [15], Heavy Edge Matching [16], Algebraic Distance [17], Affinity [18], and Kron Reduction [19], excel at deriving high-dimensional abstractions, their direct application to Boolean networks requires further adaptation. This stems from the unique structural and functional properties of Boolean networks, which differ from conventional graphs. Therefore, there is a critical need to advance these methods by incorporating a global perspective. This requires innovative strategies that harmoniously balance local and global feature learning while aligning with the inherent properties of Boolean networks.

To address these challenges, we first conduct an in-depth analysis of Boolean networks, uncovering key attributes that define their behavior: Boolean dependency, reachability, reconvergence, and the inherent tension between "static" and "dynamic" characteristics. Based on these insights, we propose *BoolSkeleton*, a novel Boolean network skeletonization method that employs homogeneous pattern reduction to balance these attributes while preserving coarse-grained information. *BoolSkeleton* consists of two primary phases: preprocessing and reduction. In preprocessing, the Boolean network is transformed into a Boolean dependency graph, with functionality-related node statuses initialized to reflect their

Manuscript received xx.xxxx.xxxx; revised xx.xxxx.xxxx; accepted xx.xxxx.xxxx. Date of publication xx.xxxx.xxxx; Date of the current version xx.xxxx.xxxx. This article was recommended by Associate Editor Testa, Eleonora. (*Corresponding author: Xingquan Li, Email: lixq01@pcl.ac.cn*)

Liwei Ni, Biwei Xie, and Huawei Li are with the State Key Lab of Processors, Institute of Computing Technology, Chinese Academy of Sciences, Beijing 100190, China. Liwei Ni is also with Pengcheng Laboratory, Shenzhen 518055, China, and also with the University of Chinese Academy of Sciences, Beijing, 101408, China.

Jiaxi Zhang is with the Peking University, Beijing, 100871, China.

Shenggen Zheng is with the Quantum Science Center of Guangdong-Hong Kong-Macao Greater Bay Area, Shenzhen,518045, China.

Junfeng Liu, Xingyu Meng, and Xingquan Li are with the Pengcheng Laboratory, Shenzhen, 518055, China.

Digital Object Identifier xx.xxxx/TCAD.xxxx.xxxxxxx.

dependencies. Then, the heterogeneous and homogeneous patterns are defined to facilitate the reduction step. Heterogeneous patterns can preserve the functionality-dependent structures, while homogeneous patterns enable node reduction to coarsen the graph. An iterative reduction algorithm, guided by the topological order of the Boolean dependency graph, is then applied to eliminate homogeneous patterns systematically. We evaluate *BoolSkeleton* across several analysis and downstream tasks: compression, classification, critical path analysis, and timing prediction. The compression analysis evaluates the network coarsening ratio; the classification analysis validates the consistency; and the critical path analysis task demonstrates its superior profiling capability. Moreover, *BoolSkeleton* achieves over 55% improvement in average accuracy compared to the original Boolean network in the timing prediction task. These experimental findings highlight the significant potential of *BoolSkeleton* in enhancing the consistency and reliability of Boolean network analysis. The contributions can be summarised as follows:

- We introduce *BoolSkeleton* to coarsen the Boolean network by leveraging the node-level homogeneous pattern reductions. To the best of our knowledge, this is the first work to study the skeleton problem on Boolean networks.
- We provide a comprehensive analysis of Boolean networks, identifying key attributes: Boolean dependency, reachability, reconvergence, and the tension between "static" and "dynamic" attributes.
- *BoolSkeleton* can well balance the local functional structure of Boolean networks with the coarse-grained skeleton, overcoming the over-reliance on the fine-grained structure of Boolean networks.
- We demonstrate the effectiveness of *BoolSkeleton* by multiple downstream tasks, achieving significant improvements in classification and timing prediction accuracy.

The rest of this paper is organized as follows: Section II provides an overview of the background and motivation; Section III elaborates on the problem statement; Section IV details the proposed *BoolSkeleton*; Section V presents experimental results for the downstream tasks; Section VI gives the discussion; and Section VII summarizes the conclusions.

II. Background and Motivation

In this section, we will introduce the background of Boolean network and the motivation of this work.

A. Background

1) Boolean Network

A Boolean network, denoted as $\mathcal{C}$, can be modeled as a computational graph which consists of functionality and the gate-based DAG. Formally, let $\mathcal{C} = (\mathcal{V}, \mathcal{E})$, where $\mathcal{V}$ is the set of nodes and $\mathcal{E}$ is the set of edges. The node set is partitioned as $\mathcal{V} = \mathcal{V}^I \cup \mathcal{V}^G \cup \mathcal{V}^O$, with $\mathcal{V}^I$ denoting the set of *Primary Input* nodes (PIs), $\mathcal{V}^O$ the set of *Primary Output* nodes (POs), and $\mathcal{V}^G$ the set of internal *logic gates*. Edges in $\mathcal{E}$ refers to the signal propagation: an edge $v_i \to v_j \in \mathcal{E}$ (where $v_i, v_j \in \mathcal{V}$) indicates that v_i is a *fanin* of v_j, and equivalently, v_j is a *fanout* of v_i.

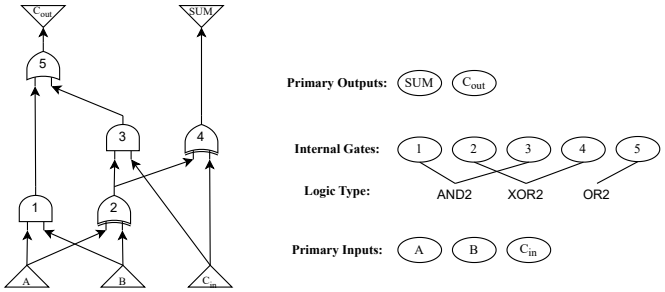
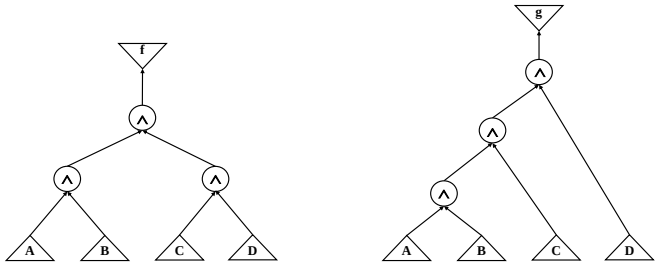


Fig. 1: The visualization of a Boolean network for a full adder, where the boolean expression of SUM and C_{out} can be formulated by "$SUM = C_{in} \oplus (A \oplus B)$, $C_{out} = \big(A \wedge B\big) \vee \big(C_{in} \wedge (A \oplus B)\big)$", respectively.



(a) $\mathcal{C}_1 : f = (A \wedge B) \wedge (C \wedge D)$ (b) $\mathcal{C}_2 : g = ((A \wedge B) \wedge C) \wedge D$

Fig. 2: The illustration of the Boolean equivalence: the Boolean expression of f in (a) and g in (b).

For a given node v_j, the set of all its fanin nodes forms its *Transitive FanIn-cone* (TFI), while the set of all its fanout nodes constitutes its *Transitive FanOut-cone* (TFO). For the DAG component of the Boolean network $\mathcal{C}$ with n nodes, we define $\mathbf{A}^{n\times n}$ as the Boolean adjacency matrix, where $\mathbf{A}_{i,j} =$ true if there exists an edge $v_i \to v_j$. Additionally, we define $\mathbf{R}^{n\times n}$ as the reachability matrix, where $\mathbf{R}_{i,j} =$ true if there exists a path from v_i to v_j. The *depth* of node $v_i \in \mathcal{V}$ is determined using the unit delay model [20]:

$$\text{depth}(v_i) = \begin{cases} 0, & v_i \in \text{PIs}, \\ \max\left(\text{depth}_{(\forall v_j \in fanin(v_i))}(v_j)\right) + 1, & \text{otherwise}, \end{cases} \tag{1}$$

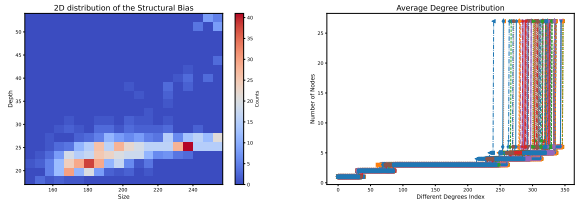
Logic gates within the Boolean network can be constructed from functionally complete sets, such as {AND2, INVERTER}, {XOR2, AND2, INVERTER}, etc. Furthermore, any superset of a functionally complete set retains functional completeness. The structure of a Boolean network representing a full adder is illustrated in Fig. 1.

2) Boolean Equivalence

The Boolean equivalence theory [3] asserts that Boolean networks with the same functionality can have different graph structures, in other words, Boolean networks with different graph structures can lead to the same functionality.

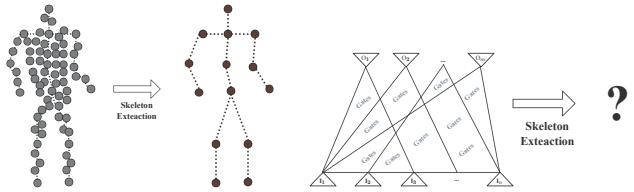
Lemma 1. *According to Boolean equivalence, given a pair of Boolean networks ($\mathcal{C}_1$, $\mathcal{C}_2$), we can say that $\mathcal{C}_1$ and $\mathcal{C}_2$ are Boolean equivalent under the following three conditions:*

$$\begin{aligned} &\textit{conditon1:} \quad \big(\mathcal{C}_1^G \equiv \mathcal{C}_2^G\big) \quad \Rightarrow \quad (\mathcal{C}_1 \doteq \mathcal{C}_2); \\ &\textit{conditon2:} \quad \big((\mathcal{C}_1^G \neq \mathcal{C}_2^G) \wedge (\mathcal{C}_1^F \equiv \mathcal{C}_2^F)\big) \quad \Rightarrow \quad (\mathcal{C}_1 \doteq \mathcal{C}_2), \end{aligned} \tag{2}$$



(a) size and depth distr. (b) average degree distr.

Fig. 3: The structural bias example of *router* design.



(a) Human Skeleton. (b) Boolean network Skeleton?

Fig. 4: The motivation of the Boolean network skeleton.

where $\doteq$ denotes as the Boolean equivalent operator, $\mathcal{C}^G$ refers to the DAG structure of Boolean network $\mathcal{C}$, and $\mathcal{C}^F$ refers to the functionality.

Lemma 1 delineates two conditions under which Boolean networks $\mathcal{C}_1$ and $\mathcal{C}_2$ are considered equivalent:

1) Identical DAG Structures: If $\mathcal{C}_1^G \equiv \mathcal{C}_2^G$, identical graph structures imply equivalent functionality, as the same topology consistently with the same node types yields the same Boolean function;
2) Distinct DAG Structures with Identical Functions: If $\mathcal{C}_1^G \neq \mathcal{C}_2^G$ yet $\mathcal{C}_1^F \equiv \mathcal{C}_2^F$, equivalence holds despite structural differences, representing a special case of Cond-1.

Fig. 2 illustrates the scenario corresponding to Condition 3 of Boolean equivalence. For instance, consider two functions f and g such that, after logic equivalence transformation, both simplify to $f = g = A \wedge B \wedge C \wedge D$, thus, $\mathcal{C}_1 \doteq \mathcal{C}_2$.

B. Motivation

1) Structural Bias

According to Boolean equivalence theory, distinct graph structures within a Boolean network can yield identical functionality, introducing structural bias [21] specific to a given design. Fig. 3 illustrates this structural bias in the design "router", demonstrating that variants differing in size, depth, or sorted node degree[1] distribution exhibit the same functional behavior. Consequently, this equivalence across variants incurs a computational overhead when verifying equivalence to assess structural bias.

2) Skeleton

In graph theory, graph reduction [22], [23] refers to a technique that reduces the number of nodes or edges in a graph while striving to preserve its fundamental structure and properties. This approach can enhance the quality and efficiency of graph-level tasks. For instance, Fig. 4 illustrates how the skeleton of a human graph simplifies pose-related tasks by abstracting the essential structure.

In logic synthesis, tasks such as verification and critical path prediction are often hindered by redundant nodes and edges, as well as structural bias. Structural bias introduces inconsistencies in graph structure, whereas graph skeletonization aims to abstract the original graph into a more concise representation. By integrating the concepts of structural bias and graph skeletonization, it is evident that an abstraction of Boolean networks is essential to improve the performance of related tasks.

C. Related Work

In graph theory, the skeleton reduction method [22] offers a robust approach to distill coarse-grained information from a graph while preserving its essential attributes. Loukas introduced a graph reduction method with spectral and cut guarantees, linking approximation quality to graph properties like degree and eigenvalue distributions [15]. Loukas et.al used heavy edge matching for spectral approximations, showing coarse eigenvectors can approximate clustering without refinement [16]. Ron et al. proposed relaxation-based coarsening using algebraic distances for multiscale graph organization [17]. Livne et al. developed the Lean Algebraic Multigrid (LAMG) method, leveraging node affinity for scalable Laplacian solving [18]. Dörfler et al. introduced Kron reduction for electrical networks, preserving connectivity in reduced graphs [19]. However, Boolean networks differ fundamentally from traditional graphs, presenting a unique challenge: how to adapt skeleton extraction techniques to Boolean networks from the perspective of logic synthesis.

III. Problem Formulation and Analysis

In this section, we will formally define the Boolean network skeleton problem. Then, we analyze the critical factors that substantially impact the efficacy of prospective solutions.

A. Problem Formulation

Definition 1 (Boolean dependency). *Given a Boolean network $\mathcal{C}$, we say that node b is Boolean dependent on node a if there exists a path from a to b in the DAG and a Boolean function f such that $b = f(a, \dots)$, and the value of b can be determined by the value of a.*

Boolean dependency is based on the concept of reachability in graph theory, a key element for comprehending the flow of information within a DAG. This dependency underscores functionality, setting it apart from conventional node dependencies in DAGs. To support graph skeletonization in this work, we introduce the following definition of a Boolean dependency graph tailored for graph operations.

Definition 2 (Boolean dependency graph). *Boolean dependency graph $\mathcal{G}$ is derived from the Boolean network, where the relationship between nodes refers to Boolean dependency.*

The Boolean network skeleton problem is based on Boolean dependency graph, and it can be described by: For any given

[1]Sorted node degree of a graph can avoid difference by calculating orders

$$
\begin{aligned}
S_0 =& A_0 \oplus B_0, \quad CO_0 = A_0 \wedge B_0, \\
S_i =& (A_i \oplus B_i) \oplus (CO_{i-1} \wedge (A_{i-1} \oplus B_{i-1})), \text{ for } i = 1, 2, 3; \\
CO_i =& (A_i \wedge B_i) \vee (CO_{i-1} \wedge (A_{i-1} \oplus B_{i-1})) \vee \\
& (A_i \wedge CO_{i-1}) \vee (B_i \wedge CO_{i-1}), \text{ for } i = 1, 2, 3.
\end{aligned} \tag{4}
$$

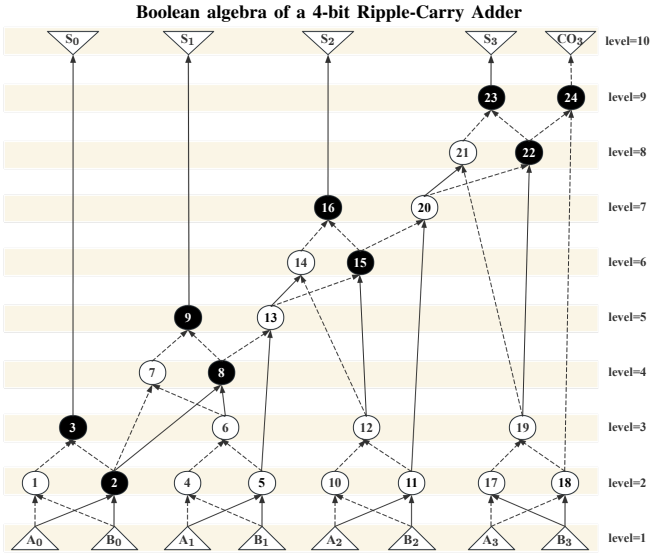


Fig. 5: The Boolean network visualization of a 4-bit ripple-carry adder in AIG format.

Boolean network $\mathcal{C}$, the Boolean skeleton problem can be defined as the reduction function **BNetworkSkeletonize** that maps $\mathcal{C} = (\mathcal{V}, \mathcal{E})$ to its corresponding reduced Boolean dependency graph $\mathcal{G} = (\mathcal{V}', \mathcal{E}')$:

$$
\begin{aligned}
\mathcal{G} \leftarrow & \textbf{BNetworkSkeletonize}(\mathcal{C}), \\
& \Rightarrow \mathcal{G} \approx \mathcal{C},
\end{aligned} \tag{3}
$$

where $|\mathcal{V}'| \leq |\mathcal{V}|, |\mathcal{E}'| \leq |\mathcal{E}|$, and the "$\approx$" symbol means $\mathcal{G}$ still retains the essential abstraction information of $\mathcal{C}$.

B. Analysis

Following the problem formulation in Eq. (3), we address the Boolean network skeleton problem by investigating the following fundamental questions from the skeleton viewpoint:

1) What information is fundamental for any given Boolean network?
2) What information can be directly extracted from a specific Boolean network?
3) What information can be simplified with minimal impact on the abstraction of a specific Boolean network?

To address the questions outlined above, we present observations derived from a typical 4-bit ripple carry adder (CRA), which subsequently inform a preliminary solution direction. For a 4-bit CRA, defined by $F[4] = A[4] + B[4]$, where S represents the 4-bit sum vector and CO denotes the 4-bit vector of carry-out bits, the Boolean function expression is provided in Eq. (4). Additionally, Fig. 5 illustrates its graph structure to enable deeper analysis. To aid this examination, we highlight the sum and carry-out nodes in the graph and assign levels to all nodes. Based on this, we derive the following observations and analysis:

Observation 1: Reachability For any node a and its transitive fanin-cone node set $\mathcal{V}^{\text{TFI}}$, all nodes in $\mathcal{V}^{\text{TFI}}$ are reachable to a. Based on Observation 1, nodes along a single path exhibit varying depths, establishing a Boolean dependency relationship. Conversely, if nodes a and b reside on distinct paths, they lack both Boolean dependency and reachability, highlighting the path-specific nature of these relationships.

Observation 2: Boolean Dependency Boolean dependency is an inherent attribute of Boolean networks, as established by its definition. It implies that the dependency between nodes influences their depth ordering under the unit delay model [20]. Specifically, if node a depends on node b, then $\mathit{depth}(a) > \mathit{depth}(b)$ must hold, reflecting the directional flow of information in the network.

Observation 3: Reconvergence Reconvergence arises as an inevitable consequence of local sharing induced by logic optimization. It occurs when signals diverge at a node and subsequently reconverge at a later transitive fanout stage, forming a reconvergence cone. This phenomenon generates multiple parallel paths within the cone, complicating the network's structure and dependency analysis. From a graph reduction viewpoint, there exist opportunities to merge the reconvergence nodes to simplify Boolean networks.

Observation 4: Dynamic vs. Static Properties Boolean equivalence and structural bias underscore the dynamic nature of a Boolean network's structure for a specific design. Locally, Fig. 2 exemplifies this by depicting distinct graph structures for the function $F = A \wedge B \wedge C \wedge D$ under two equivalent conditions. Globally, logic optimization techniques can iteratively transform an entire Boolean network by substituting local substructures with their Boolean equivalents. From both Boolean algebra and graph-theoretic perspectives, the PIs and POs remain fixed for a given design, anchoring their static functionality within dynamic structural variations.

For instance, consider node n_9 in Fig. 5, with its transitive fanin-cone node set $\mathcal{V}^{\text{TFI}} = \{n_{A_0}, n_{B_0}, n_{A_1}, n_{B_1}, n_2, n_4, n_5, n_6, n_7, n_8\}$. The depth of each node in $\mathcal{V}^{\text{TFI}}$ is less than that of node 9, consistent with Observation 1. Reconvergence is evident as signals diverge at node 6 and merge at node 9, positioning nodes 7 and 8 on parallel paths. Consequently, nodes 7 and 8 exhibit neither Boolean dependency nor reachability, aligning with Observations 2 and 3. These insights, drawn from the intricate interplay of structure and functionality in Boolean networks, directly address the questions posed earlier. The principle of "less is more" suggests that a balanced consideration of both graph structure and functionality can yield more reliable skeletonization outcomes. This analysis diverges from traditional approaches, which often neglect functionality-related structures, underscoring the critical role of such information in effective skeleton extraction.

IV. *BoolSkeleton*: Boolean Network Skeleton

In this section, we will introduce *BoolSkeleton*, the Boolean network skeleton method as shown in Fig. 6. It comprises two primary stages: preprocessing and reduction. First, the Boolean network is transformed into a Boolean dependency graph with

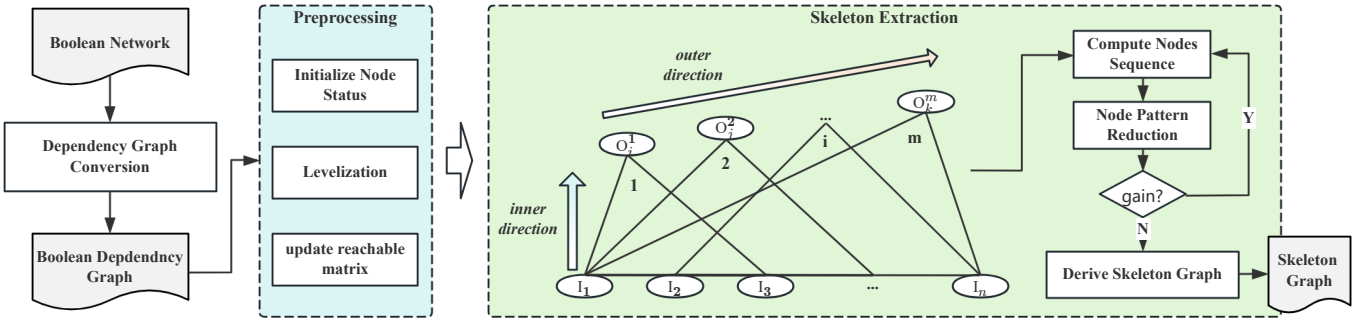


Fig. 6: The framework of the proposed Boolean Network Skeleton method, *BoolSkeleton*.

functionality-related node status assignment. Then, an iterative node-level pattern reduction approach is employed to get the reduced skeleton.

A. Phase1: Preprocessing

1) Boolean Dependency Graph Recovery

To facilitate the graph operation, the Boolean network $\mathcal{C}$ is first transformed into its corresponding Boolean dependency graph $\mathcal{G}$, a process that systematically maps the network's functional dependencies into a graph representation. To fully capture the graph structure of $\mathcal{C}$, we explicitly represent each inverter which is embedded in the edges as a distinct node in $\mathcal{G}$. This representation allows the skeletonization algorithm to determine which nodes should be reduced or merged in subsequent steps. By adopting this approach, all structural details of the Boolean network are preserved while enhancing flexibility for further manipulation.

2) Node Status Assignment

Based on the analysis in Section III-B, PIs and POs constitute critical components of a Boolean network, both from the perspectives of Boolean algebra and graph structure. In contrast, internal gates exhibit a dynamic status within the graph structure due to structural bias, rendering them more amenable to reduction or merging during Boolean network skeletonization.

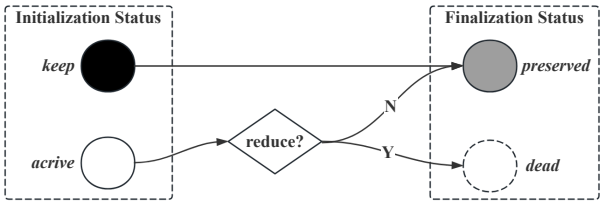
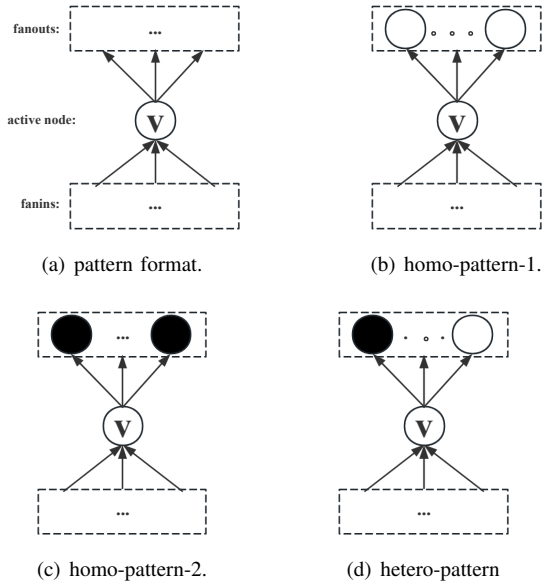


Fig. 7: Node status transformation diagram.

To differentiate between static and dynamic information, we define and assign a status to each node, categorized into four types: $\{keep, active, preserved, dead\}$. These statuses are illustrated in Fig. 7 via a transformation diagram. Initially, nodes are classified as either *keep* or *active*: *keep* designates nodes with static information that must be retained, while *active* indicates nodes with dynamic information that are candidates for reduction. Upon completion of the process, nodes transition to either *preserved* or *dead*: *preserved* signifies nodes retained in the skeleton, and *dead* denotes those eliminated. During the initialization phase, PIs and POs are assigned the *keep* status, whereas internal gates are designated as *active*:

$$\begin{aligned} v_i^s &\leftarrow keep, \quad \forall v_i \in \mathcal{V}^I; \\ v_o^s &\leftarrow keep, \quad \forall v_o \in \mathcal{V}^O; \\ v_g^s &\leftarrow active, \quad \forall v_g \in \mathcal{V}^G, \end{aligned} \tag{5}$$

In subsequent steps, *active* nodes are progressively reassigned to either *preserved* or *dead* based on the reduction process.



(a) pattern format. (b) homo-pattern-1.

(c) homo-pattern-2. (d) hetero-pattern

Fig. 8: The illustration of the defined patterns.

B. Phase2: Reduction

The reduction process leverages homogeneous pattern reduction applied to the Boolean dependency graph. We begin by defining the patterns and their corresponding reduction rules. Subsequently, an iterative, fanin-limited, node-level pattern reduction approach is employed to eliminate nodes while preserving the skeleton structure.

1) Patterns and the Rule

Definition 3 (Pattern). *A pattern* $\mathbf{P} = (v, \mathcal{V}^{fanin}, \mathcal{V}^{fanout})$ *is defined as a node-level DAG subgraph comprising three components: a central node* v*, its in-degree nodes* $\mathcal{V}^{fanin}$*, and its out-degree nodes* $\mathcal{V}^{fanout}$*.*

Definition 3 establishes the node-level pattern structure, with its graphical representation depicted in Fig. 8 (a). The

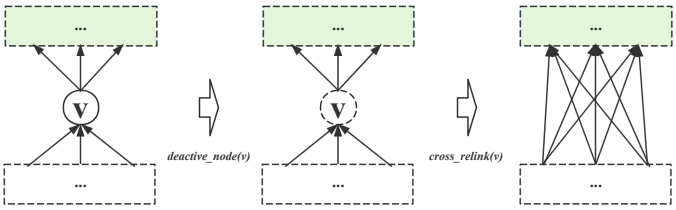


Fig. 9: The pattern reduction rule.

proposed method targets node-level reduction to derive an abstraction of the original graph, and accordingly, the defined pattern format emphasizes a node-centric perspective. In practice, patterns are classified into two categories based on the consistency of the statuses of $\mathcal{V}^{\text{fanout}}$ in $\mathbf{P}$: heterogeneous and homogeneous patterns, as detailed below:

Definition 4 (Heterogeneous Pattern). *A pattern* $\mathbf{P}$ *is termed a heterogeneous pattern, denoted* $\mathbf{P}_{hetero}$*, if its fanout nodes* $\mathcal{V}^{fanout}$ *exhibit distinct statuses.*

Definition 5 (Homogeneous Pattern). *A pattern* $\mathbf{P}$ *is termed a homogeneous pattern, denoted* $\mathbf{P}_{homo}$*, if its fanout nodes* $\mathcal{V}^{fanout}$ *share the same status.*

TABLE I: The occurrence probability of the defined patterns.

Order \ Pattern	$\mathbf{P}^1_{homo}$	$\mathbf{P}^2_{homo}$	$\mathbf{P}_{hetero}$
Topo-order (PI → PO)	high	low	low
Reverse Topo-order (PO → PI)	low	high	high

Definition 6 (Reduction rule). *The reduction rule* σ *is defined as a mapping function that maps the pattern to its target subgraph* $\mathbf{P}'$*:*

$$\mathbf{P}' \leftarrow \sigma(\mathbf{P}), \tag{6}$$

where $\mathbf{P} = v \cup \mathcal{V}^{in} \cup \mathcal{V}^{out}$*,* $\mathbf{P}' = \mathcal{V}^{in} \cup \mathcal{V}^{out}$*,* $\mathbf{P}^{'} \subset \mathbf{P}$ *and* $|\mathbf{P}^{'}| = |\mathbf{P}| - 1$*.*

Fig. 8(b) and Fig. 8(c) show the homogeneous patterns P^1_{homo} and P^2_{homo}, respectively, and Fig. 8(d) shows the heterogeneous pattern P_{hetero}. Table I demonstrates the probability of the occurrence of the defined patterns according to the assigned node status. It should be noted that not all the patterns are used for the reduction. The heterogeneous pattern can distinguish the Boolean dependency between different nodes, while the homogeneous pattern can not; it mainly propagates the signals to the nodes with the same status. These properties are discussed in Property. 1 and Property. 2. It should be noted that the pattern reduction rule σ is focused on node-level reduction as shown in Fig. 9. The **PatternReduction** is defined as the node reduction operator on an "active" node v of the Boolean dependency graph $\mathcal{G}$ by the reduction rule σ:

$$\textbf{PatternReduction}\Big(\mathcal{G}, v, \boldsymbol{A}, \boldsymbol{R}, \mathbf{P}, \sigma(\mathbf{P}), K\Big), \tag{7}$$

where K is the limitation of the fanin size of a pattern, which can be used to control the graph coarsening ratio.

Algorithm 1 Iterative Node-level Pattern Reduction

Input: Boolean dependency graph $\mathcal{G}$, homogeneous pattern $\mathbf{P}_{homo}$, reduction rule σ, limitation K
Output: Skeleton graph $\mathcal{G}^{'}$

$\boldsymbol{A}, \boldsymbol{R} \leftarrow$ update_graph_matrix(G)
levelization$(\mathcal{G})$
initialize_node_status$(\mathcal{G})$
$\mathcal{V}^{PO'} \leftarrow$ sort_nodes_by_level_ascending$(\mathcal{V}^{PO})$
while *true* **do**
  set $count \leftarrow 0$
  for v_o in $\mathcal{V}^{PO'}$ **do**
    $\mathcal{V}^{fanincone} \leftarrow$ collect_fanincone_by_dfs$(\mathcal{G}, v_o)$
    for v in $\mathcal{V}^{fanincone}$ **do**
      if get_fanin_size$(\mathcal{G}, v) \geq K$ **then**
        set_node_status_preserved$(\mathcal{G}, v)$
        continue
      end if
      $count$ += **PatternReduction**$(\mathcal{G}, v, \boldsymbol{A}, \boldsymbol{R}, \mathbf{P}_{homo}, \sigma)$
    end for
  end for
  if $count = 0$ **then**
    break
  end if
end while
$\mathcal{G}^{'} \leftarrow$ skeleton_nodes_collection$(\mathcal{G})$
Return: $\mathcal{G}^{'}$

For any given Boolean dependency graph $\mathcal{G}$ with the adjacent matrix $\mathbf{A}$ and the reachable matrix R, current processed node v_k, the pattern reduction can be formulated:

$$v_k^s \leftarrow dead,$$
$$\forall v_i \in \mathcal{V}^{fanin}_{v_k}, \forall v_o \in \mathcal{V}^{fanout}_{v_k}. \begin{cases} 1.\ \boldsymbol{A}_{v_i,v_k} \leftarrow 0, \\ 2.\ \boldsymbol{A}_{v_k,v_o} \leftarrow 0, \\ 3.\ \boldsymbol{A}_{v_i,v_o} \leftarrow 1\ (\boldsymbol{A}_{v_i,v_o} = 0), \\ 4.\ \boldsymbol{R}_{v_i,v_o} \leftarrow 1, \end{cases} \tag{8}$$

As illustrated by Fig. 9 and Eq. (8), the process begins by updating the status of node v_k to *dead* and severing all its connections. Subsequently, for each pair of nodes from $\mathcal{V}^{\text{fanin}}_{v_k}$ to $\mathcal{V}^{\text{fanout}}_{v_k}$, a direct edge is established ($\mathbf{A}_{v_i,v_o} = 1$) if no prior path existed—where a path implies reachability—and the reachability matrix is updated accordingly ($\mathbf{R}_{v_i,v_o} = 1$).

2) Iterative Node-level Pattern Reduction Approach

The Boolean dependency graph reduction can be addressed by the iteratively fanin-limited node-level homogeneous pattern reduction:

$$\mathcal{G} \leftarrow \textbf{BNetworkSkeletonize}(\mathcal{C}),$$
$$\Rightarrow \mathcal{G}^{'} \leftarrow \textbf{LimitedPatternReduction} \overset{n-1}{\underset{i=0}{}} \left(\mathcal{G}, v_i, \boldsymbol{A}, \boldsymbol{R}, \mathbf{P}_{homo}, \sigma, K\right) \tag{9}$$

Eq. (9) provides a detailed formulation of the Boolean network skeleton solution through an iterative, node-level, fanin-limited pattern reduction approach. Algorithm 1 takes inputs of the converted Boolean dependency graph $\mathcal{G}$, the homogeneous pattern $\mathbf{P}_{\text{homo}}$, the reduction rule σ, and a fanin size limit K, producing the extracted skeleton graph $\mathcal{G}'$ as output. The process begins with a preprocessing phase

Algorithm 2 Try Homogeneous Pattern Reduction

Input: Boolean Dependency Graph $\mathcal{G}$, adjacent matrix $\boldsymbol{A}$, reachable matrix $\boldsymbol{R}$, node v, pattern $\mathbf{P}$, reduction rule σ
Output: The reassigned dead node count: $count$

1: Initialize $count \leftarrow 0$
2: **if** not is_node_status_activate($\mathcal{G}$, v) **then**
3: **Return:** $count$
4: **end if**
5: **if** is_match_pattern($\mathcal{G}, v, \mathbf{P}$) **then**
6: set_node_status_dead($\mathcal{G}, v$)
7: reset_adjacent_matrix_at($v, \boldsymbol{A}$)
8: reset_reachable_matrix_at($v, \boldsymbol{R}$)
9: add_cross_edges_and_reachability($v, \boldsymbol{A}, \boldsymbol{R}$)
10: $count \leftarrow 1$
11: **else**
12: set_node_status_preserved($\mathcal{G}, v$)
13: **end if**
14: **Return:** $count$

(lines 1–3), which involves computing the adjacency matrix $\mathbf{A}$ and reachability matrix $\mathbf{R}$, updating each node's level using the unit delay model [20], and initializing node statuses as Eq. (5). Subsequent steps focus on node-level reduction. Initially, primary output nodes are collected and sorted into $\mathcal{V}^{\text{PO}'}$ in ascending order of depth. The unit delay model ensures that processing nodes in depth order preserves Boolean dependency. Within the while loop, a counter $count$ tracks the number of nodes reassigned to *dead* in each iteration. The fanin-limited pattern reduction is applied in topological order to the transitive fanin-cone $\mathcal{V}^{\text{faninconе}}$ of a specific primary output node. If a node's fanin size exceeds K, its status is set to *preserved* (lines 9–14); otherwise, it is evaluated for reduction via the *try_node_pattern_reduction*() function, detailed in Algorithm 2 (line 15). The loop terminates when no nodes are reduced in an iteration (lines 18–19). Finally, the skeleton graph $\mathcal{G}'$ is constructed by removing all *dead* nodes and connecting the remaining *preserved* nodes of $\mathcal{G}$ (line 22).

Algorithm 2 elaborates the homogeneous pattern reduction process, adhering to Eq. (8), with its application context illustrated in Algorithm 1 (line 15). The variable $count$ indicates whether the current node is reassigned to *dead*, returning 1 for true and 0 for false. The procedure first verifies that the node v's status is *active*; if not, it exits immediately. If node v and its surrounding subgraph match the specified pattern, a node-level reduction is executed following the four steps in Eq. (8) (lines 6–9). Notably, an edge between v's fanin and fanout nodes is added only if no prior reachability exists, preventing the introduction of redundant or nested edges that could compromise the skeleton graph's integrity.

3) Theorem Analysis

Based on the definitions of the proposed patterns and their reduction rule, we derive several key properties that echo the Boolean network analysis in Section III-B.

Property 1. *The homogeneous pattern $\mathbf{P}_{homo}$ does not alter the depth order among primary output (PO) nodes. As depicted in Fig. 8 (b) and (c), all fanout nodes share the same status. Per Property 2, the central node v_c does not contribute to distinguishing its fanout cones, thus preserving the relative depth order of POs.*

Property 2. *The heterogeneous pattern $\mathbf{P}_{hetero}$ preserves the depth order among PO nodes. As illustrated in Fig. 8 (d), a heterogeneous pattern $\mathbf{P}_{hetero}$ includes at least one keep node $v_k \in \mathcal{V}^{fanout}$ and one active node $v_a \in \mathcal{V}^{fanout}$. Assuming $depth(v_c) = d_c$ for the central node v_c, and excluding the influence of other nodes on this pattern's depth, we have $depth(v_k) = d_c + 1$ and $depth(v_a) = d_c + 1$. If v_a targets a PO node v_{po}, then,according to unit delay model in Eq. (1), $depth(v_{po}) = \max_{(v_i \to v_{po}) \in \mathcal{E}}(depth(v_i) + 1)$. Thus, $depth(v_{po}) \geq depth(v_a) + 1 = d_c + 2 > depth(v_k)$, ensuring the depth order is maintained.*

Proposition 1. *The **PatternReduction** operator preserves both the reachability relations and topological order of the remaining nodes in the Boolean dependency graph $\mathcal{G}$. Additionally, this preserved reachability ensures the original fanout propagation across the transitive fanin (TFI) and fanout (TFO) cones of the reduced node.*

Proof 1 (Proof of Proposition 1). *Consider a node v, for topological order, the transitive property of partial ordering states that if $v_i \preceq v$ and $v \preceq v_o$, then $v_i \preceq v_o$. This is verified as follows:*

$$\begin{aligned} &\forall v_i \in \mathcal{V}_v^{fanin},\ v_i \preceq v, \\ &\forall v_o \in \mathcal{V}_v^{fanout},\ v \preceq v_o, \\ &\Rightarrow \forall v_i \in \mathcal{V}_v^{fanin}, \forall v_o \in \mathcal{V}_v^{fanout},\ v_i \preceq v_o. \end{aligned}$$

*Thus, iterative applications of **PatternReduction** preserve both reachability and topological order for all retained nodes.*

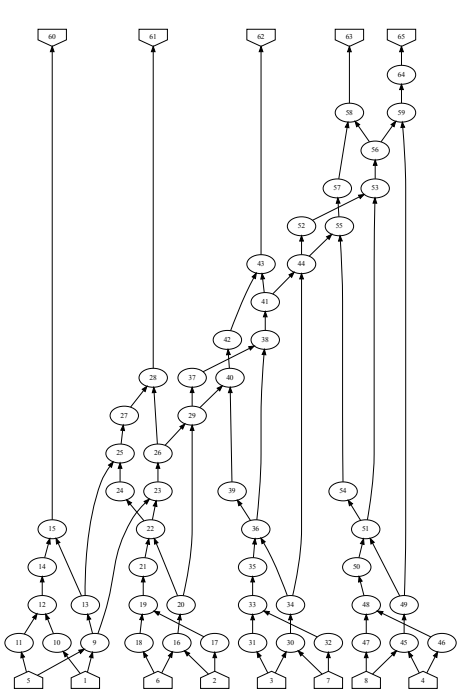
We conclude that the **PatternReduction** operator is both reachability- and topology-aware, making it suitable for applications requiring these properties.

C. Case Study

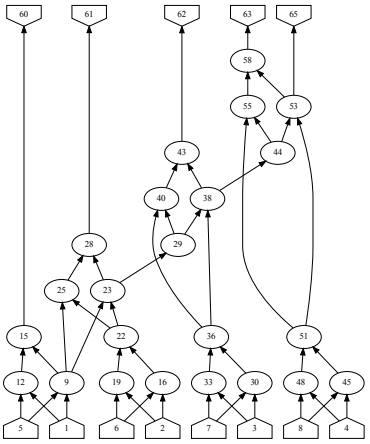
Fig. 10 presents a case study of the Boolean network skeleton applied to a 4-bit ripple carry adder design, as shown in Fig. 5, under different coarsening ratios by the fanin size limitation K of the pattern. Specifically, Fig. 10 (a) depicts the Boolean dependency graph directly derived from the initial Boolean network (AIG). This case study yields two primary observations:

1) As the fanin constraint K increases, the graph coarsening ratio rises, reducing the number of nodes and yielding a more pronounced skeleton structure.
2) The similarity between the skeleton and the original Boolean dependency graph decreases, as the retained information diminishes with a greater coarsening ratio.

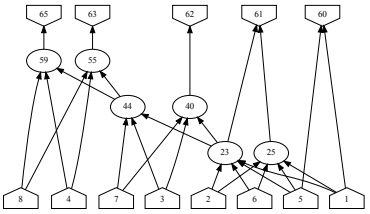
These observations highlight a key insight: the fanin constraint K in *BoolSkeleton* acts as a control parameter for the graph coarsening ratio. The optimal coarsening level varies across tasks; for instance, functionality-related tasks prioritize preserving heterogeneous structures, while Boolean network profiling tasks emphasize balancing local information with the



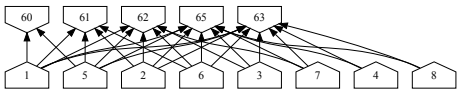
(a) Skeleton (K=1) [a]



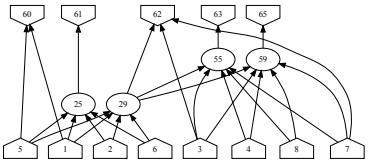
(b) Skeleton (K=2)



(c) Skeleton (K=3)



(e) Skeleton (K=∞ (unlimited))



(d) Skeleton (K=4)

[a]K=1 means the Boolean network is first translated into a Boolean dependency graph.

Fig. 10: The case study of the skeleton with different fanin limitation of the 4-bit ripple carry adder in Fig. 5.

global skeleton. To assess the divergence between the skeleton and the original Boolean dependency graph as K increases, graph similarity metrics, such as those based on spectral analysis [24], [25], provide a robust evaluation tool. Selecting an appropriate K thus depends on task-specific requirements, and while a universal value remains elusive, such metrics enable tailored similarity assessments. Further exploration of this variability is deferred to subsequent task-specific analyses.

V. Empirical Evaluation

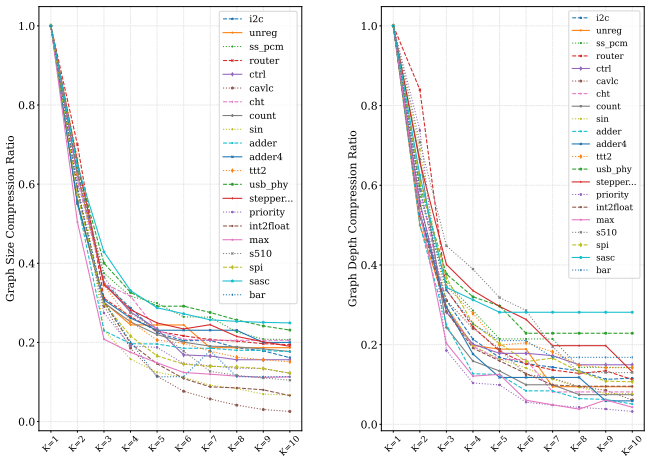
In this section, we will show the effectiveness of *BoolSkeleton* by conducting two validation evaluations: compression and classification, and two downstream tasks: critical path analysis and timing prediction.

A. Setup

Environment. The codes of *BoolSkeleton* are written in C++, and the following tasks are conducted in Python. The experiments were conducted using the following hardware and software configuration: *(Hardware)* Intel Xeon Platinum 8380 CPU (160 cores), 512 GB RAM, NVIDIA A100 GPU (40 GB VRAM); *(Software)* Ubuntu 20.04.6, Python 3.8, PyTorch 2.0.1, CUDA 12.0, torch_geometry 2.3.1, scikit-learn 1.2.2, pandas 1.5.3, matplotlib 3.7.1. This high-performance setup ensures efficient processing of large datasets and complex computations, providing a reliable foundation for experimental evaluation.

Dataset. The dataset consists of benchmarks widely adopted in logic synthesis, sourced from IWLS2005 [26] and IWLS2015 [27]. Table II summarizes the characteristics of the dataset, including the number of primary inputs (#PI), primary outputs (#PO), gates (#Gate), inverters (#Inverter), two-input AND gates (#AND2), edges (#Edge), and circuit depth (Depth). The dataset's variety—spanning small-scale designs like *ctrl* to complex ones like *sin*—enables robust evaluation of our approach. This selection ensures diversity across the downstream tasks, offering a comprehensive testbed for *BoolSkeleton*.

Baseline. Baseline methods are from a diverse set of techniques, including variation-based approaches [15] ('variation_neighborhoods', 'variation_edges', 'variation_cliques'), edge-weight optimization [16] ('heavy_edge'), algebraic methods [17] ('algebraic_JC'), affinity-guided strategies [18] ('affinity_GS'), and Kronecker-based reduction [19] ('kron'). These methods, implemented using [28], [15], are configured with a uniform reduction ratio of 0.3, and compared to assess their effectiveness in reducing graph complexity while preserving structural properties for downstream Graph Neural



(a) Graph Size (b) Graph Depth

Fig. 11: Compression ratio over the graph size and depth.

Network tasks.

TABLE II: The characteristics of the source designs.

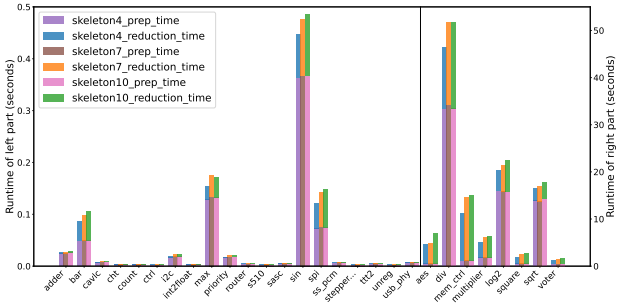
Design \ Char	#PI	#PO	#Gate	#Inverter	#AND2	#Edge	Depth
adder	256	129	2547	1527	1020	2172	511
bar	135	128	6928	3592	3336	6928	21
cavlc	10	11	1606	913	693	1400	33
cht	47	36	595	324	271	614	17
count	35	16	467	275	192	416	40
ctrl	7	26	419	245	174	380	20
i2c	147	142	2785	1443	1342	2859	36
int2float	11	7	545	285	260	533	31
max	512	130	6366	3501	2865	5988	495
priority	128	8	2350	1372	978	1970	499
router	60	30	491	234	257	545	73
s510	25	15	432	211	221	470	15
sasc	113	63	692	337	355	809	16
sin	24	25	11479	6063	5416	10879	350
spi	240	239	8037	4250	3787	8049	63
ss_pcm	104	90	829	426	403	932	14
steppermotordrive	28	27	330	150	180	395	16
ttt2	24	21	793	442	351	730	28
unreg	36	16	411	253	158	348	16
usb_phy	132	90	963	476	487	1101	16
aes	683	529	51173	22518	28655	115418	44
div	128	128	64826	37726	27100	108555	8406
mem_ctrl	1187	962	20324	10323	10001	41691	58
multiplier	128	128	58958	31205	27753	111242	524
log2	32	32	68409	36027	32382	129590	597
square	64	128	43595	24096	19499	78151	445
sqrt	128	64	78246	45647	32599	130518	10384
vouter	1001	1	24736	14208	10528	42114	113

B. Validity evaluation

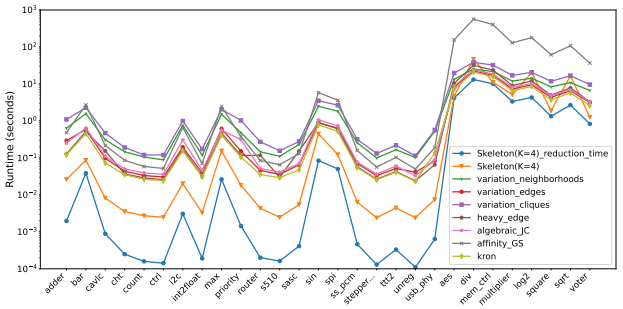
1) Analysis 1: Boolean Network Compression

Fig. 11 illustrates the compression ratios achieved across selected designs with K from 1 to 10;. This diagram reveals that increases in the fanin constraint K correspond to reductions in both graph size and depth. Specifically, a larger K permits greater merging of local nodes, enhancing the compression effect.

Fig. 12(a) illustrates the runtime performance of Boolean skeleton methods for $K = 4, 7$, and 10 across the selected designs. Each bar is composed of preparation time and node reduction time, highlighting that preparation time constitutes



(a) Runtime of different Boolean network skeletons.



(b) Runtime comparison between different methods.

Fig. 12: Runtime analysis.

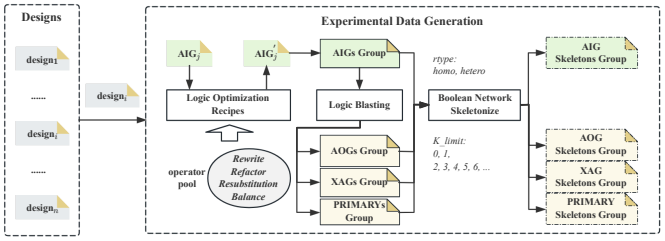


Fig. 13: Boolean network classification dataset flow.

the predominant portion of the total runtime. Additionally, the runtime for a given design remains relatively consistent across the different K values. Fig. 12(b) presents a runtime comparison between the Boolean skeleton methods with $K = 4$ and the baseline graph coarsening methods. This comparison demonstrates that the Boolean skeleton method consistently outperforms all baseline approaches in terms of runtime.

2) Analysis 2: Boolean Network Classification

The Boolean network classification task aims to evaluate the efficiency and robustness of the proposed skeleton method within the context of Boolean network analysis. The criterion for classification is that Boolean networks with the same functionality belong to the same class. This task examines two primary aspects: (1) the extent to which the skeleton enhances computational efficiency for a specific Boolean network type, and (2) the method's ability to maintain consistent classification performance across diverse Boolean network representations of a given design.

Dataset Generation Flow. Fig. 13 shows the Boolean network classification dataset generation flow. For each design, the process begins by loading it into an And-Inverter Graph

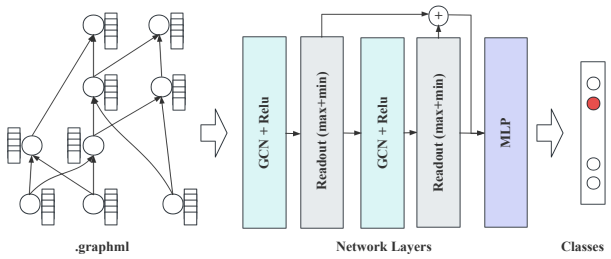
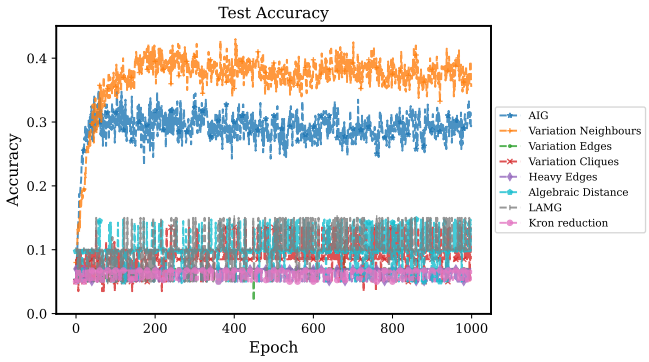


Fig. 14: The GCN-based model of the graph classification.



(a) Comparison between different methods over AIGs.

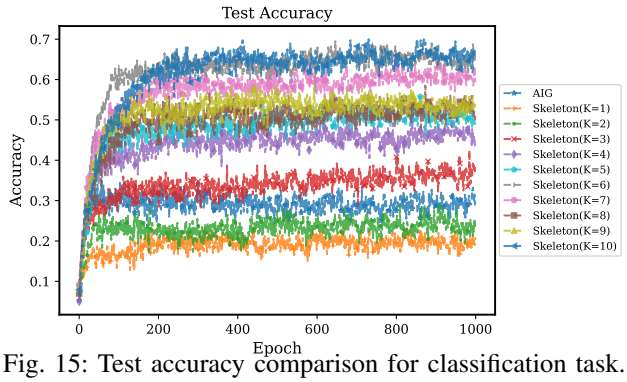


Fig. 15: Test accuracy comparison for classification task.
(b) Comparison between *BoolSkeleton* with different K.

(AIG)-based Boolean network. Subsequently, logic optimization techniques are applied to produce a diverse set of AIG variants with a size of 1,000. Each optimization sequence is randomly and redundantly generated by the operators in {rewrite, refactor, balance, resub} from berkeley-abc [29] tool with a max size of 10. These variants are then converted into other types of Boolean networks by logic blasting [30]. Finally, the proposed Boolean network skeleton methods with different K are employed to extract the corresponding skeleton from each network.

GCN-based Model. The graph convolutional network (GCN) architecture employed for classification is depicted in Fig. 14. The input consists of DAGs in *GraphML* format for the *torch_geometry* [31] package, augmented with node features representing circuit properties (node types by one-hot embedding). This model comprises *"GCN + ReLU"* layers, followed by an graph embedding *"Readout"* stage, and concludes with an *"MLP + Softmax"* layer to predict its class label. The training process spans 1,000 epochs, with a learning rate of 0.001, and employs cross-entropy loss as the loss function.

Algorithm 3 Similarity Computation Between the Boolean Dependency Graph and Netlist

Input: Boolean Network $\mathcal{C}^{logic}$, Boolean Dependency Graph $\mathcal{G}$, Gate-level Netlist $\mathcal{C}^{net}$
Output: The similarity α
1: $path_{critical} \leftarrow$ compute_critical_path($\mathcal{G}$)
2: $pathes_{topk} \leftarrow$ compute_topk_timing_path($\mathcal{C}^{net}, k = 3$)
3: $region_1 \leftarrow$ extract_critical_region($\mathcal{C}^{logic}, [path_{critical}]$)
4: $region_2 \leftarrow$ extract_critical_region($\mathcal{C}^{logic}, pathes_{topk}$)
5: $\alpha \leftarrow$ compute_similarity($region_1, region_2$)
6: **Return:** α

Evaluation. We conduct this experiment across two dimensions: (1) efficiency improvements for a specific Boolean network type, and (2) consistency across heterogeneous Boolean network representations.

Evaluation 1: The efficiency of the skeleton for one specific Boolean network type. Fig. 15(a) shows that the test accuracies of the compared baseline are worse than the original AIG dataset (which means do nothing), excluding the "Variation neighbors" method. Fig. 15(b) illustrates the test accuracy of AIG classification across different fanin constraints (K). The results highlight how *BoolSkelegon* improves generalization, with different K values striking a balance between complexity reduction and preservation of functionality-related features, as evidenced by elevated accuracy scores.

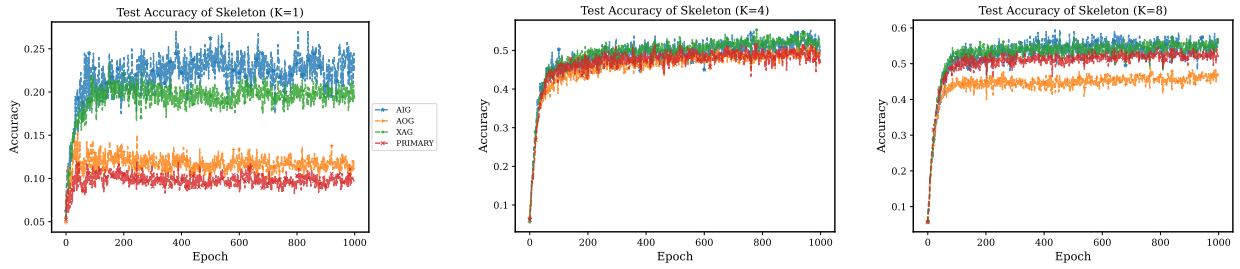
Evaluation 2: The consistency of the skeleton across different types of Boolean network. This evaluation examines the robustness of the skeleton method across varied Boolean network representations, a critical factor for EDA applications where circuits may be expressed in multiple forms (AIG, AOG, XAG, PRIMARI[2]). Fig. 16 compares classification accuracy for AIG-trained models applied to skeletonized networks with homogeneous pattern reductions and fanin limits ranging from $K = 1$ to $K = 10$. The consistent accuracy across these representations, despite structural differences, underscores the method's adaptability. This resilience to variability enhances its potential for broad adoption in EDA workflows, where maintaining functional consistency across diverse circuit models is paramount.

C. Task 1: Critical Path Analysis

In this task, we evaluate the similarity between the critical path[3] of the original Boolean network and that of its skeleton graph following technology mapping.

Similarity Computation. Algorithm 3 delineates the procedure for computing the similarity between the Boolean dependency graph and the gate-level netlist. The core idea is to map the critical path of the Boolean dependency graph and the top-3 timing paths of the gate-level netlist onto the original

[2]PRIMARY network consists of the basic primary logic gates.
[3]The critical path is defined as the path with the maximum arrival time, as determined by the static timing analysis tool iSTA [32].



(a) Skeleton(K=1) (b) Skeleton(K=4) (c) Skeleton(K=8)

Fig. 16: Test accuracy comparison of the different Boolean network types while training with AIG.

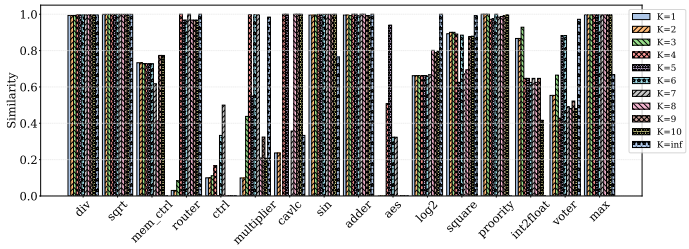


Fig. 17: The similarity comparison between the skeleton graph with different constraints (K) and the gate-level netlist.

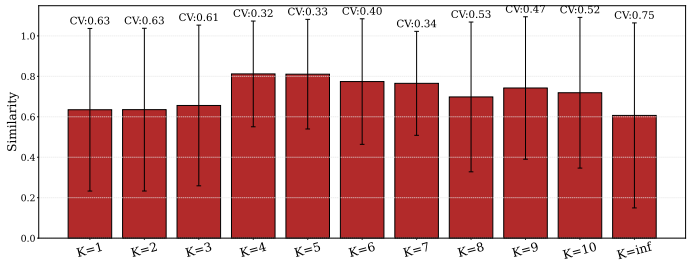


Fig. 18: Comparison of average similarity (AVE) with standard deviation (STD, represented by error bars) and coefficient of variation (CV ($CV = \frac{AVE}{STD}$), values displayed above bars) across different methods.

Boolean network (lines 1–2). The critical regions associated with these paths are extracted using a two-stage labeling technique: (1) a top-down traversal assigns label a, followed by (2) a bottom-up traversal assigns label b to the labeled nodes (lines 3–4). The similarity score α is then calculated as the overlap between these two mapped regions (line 5). A higher α indicates greater similarity between the critical paths of the skeleton graph and the netlist, suggesting that the *BoolSkeleton* usually more effectively captures the critical path characteristics compared to the original Boolean network.

Evaluation. Fig. 17 presents a similarity comparison between skeleton graphs, generated with varying fanin constraints (K), and the gate-level netlist for multiple designs. Most designs exhibit high similarity across different K values, indicating robust critical path preservation. However, certain designs—such as *router*, *ctrl*, and *aes*—display varied similarity distributions, suggesting that critical path fidelity depends on an optimal balance of information retention. Excessive or insufficient skeletonization can degrade the representation of timing-critical paths.

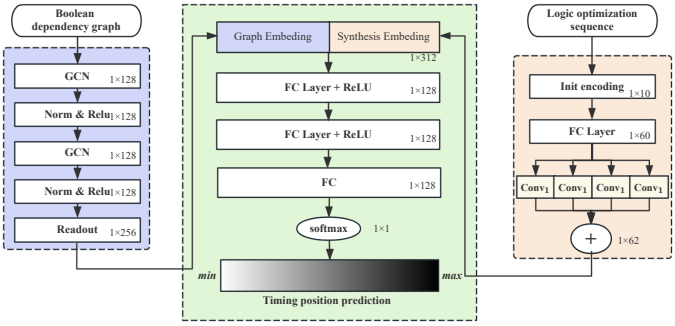


Fig. 19: GCN-based timing prediction model.

According to the case study in Section IV-C, the Boolean network classification task in Section V-B.2, and the results in Fig. 17, we infer the following: when the node count is excessively high, redundant paths emerge, complicating timing analysis; conversely, when the graph size is too low, significant information loss relative to the original circuit causes the critical path to deviate, increasing reliance on the original Boolean network for accurate timing prediction.

Fig. 18 depicts the average similarity (AVE) with standard deviation (STD, represented by error bars) and coefficient of variation (CV, defined as $CV = \frac{\text{STD}}{\text{AVE}}$, with values annotated above bars) across different K values for the skeleton method. The results indicate that $K = 4$ and $K = 5$ yield superior performance across the test designs, balancing similarity and stability. From a cut enumeration perspective, a 4-feasible cut aligns closely with optimization and mapping strategies in technology mapping, explaining the method's effectiveness.

D. Task 2: Timing Prediction

Quality of Results (QoR) prediction tasks [35], [36], [37] are garnering growing attention, as they play a pivotal role in steering the optimization process within logic synthesis. As highlighted by the critical path analysis tasks in Section V-C, the skeleton graph appears to offer a more effective representation of timing-critical paths compared to the original Boolean network. To further assess the efficacy of our approaches, we undertake a timing prediction task.

Dataset. The timing prediction dataset is constructed based on the classification dataset above. Building on this foundation, the Static Timing Analysis (STA) tool, iSTA [32], was

TABLE III: The MAPE results comparison of the Timing Prediction task.

Design \ Graph	AIG	Variation neighbors [15]	Alberaic Distance [17]	Kron reduction [19]	BoolSkeleton (*K=3*)	BoolSkeleton (*K=4*)	BoolSkeleton (*K=5*)	BoolSkeleton (*K=6*)	STD.
adder	6.578	0.509	2.090	0.668	0.524	**0.004**	*0.351*	2.891	2.202
cavlc	1.612	*1.398*	5.155	2.676	4.801	1.958	3.374	**0.861**	1.590
cht	*4.937*	22.078	26.119	7.290	10.533	**3.014**	23.440	22.253	9.426
count	0.924	8.424	9.946	4.601	*0.691*	1.780	1.262	**0.091**	3.801
ctrl	5.524	2.343	15.126	3.881	*1.887*	2.330	2.742	**1.181**	4.544
i2c	1.882	3.868	1.620	2.022	*0.053*	1.958	**0.048**	1.947	1.219
int2float	2.407	3.153	*0.307*	7.273	19.178	0.650	**0.159**	1.521	6.428
max	0.597	2.603	2.734	1.280	**0.006**	0.158	0.152	*0.013*	1.145
priority	*1.204*	3.266	3.466	11.789	3.153	**0.881**	2.268	3.405	3.429
router	5.011	**0.167**	*0.642*	6.195	14.029	9.426	8.220	2.243	4.758
s510	2.108	3.056	13.072	4.244	16.001	0.483	**0.011**	0.982	6.098
sasc	0.635	2.465	12.354	0.475	1.648	0.961	*0.436*	**0.209**	4.092
sin	**0.522**	2.483	*1.129*	1.873	1.376	2.065	1.992	2.695	0.721
spi	18.129	*0.587*	2.552	5.424	2.358	4.915	**0.033**	21.370	8.178
stepper	15.392	14.050	14.324	12.001	**1.400**	5.857	3.549	*2.997*	5.811
ttt2	22.885	10.560	**0.634**	38.760	12.759	4.346	*3.033*	31.543	14.070
unreg	1.908	7.607	33.813	13.733	1.506	**0.456**	14.381	*0.645*	11.450
usb_phy	8.607	**1.991**	9.031	4.384	8.428	3.817	*2.866*	5.115	2.782
Train time (secs).	21896	1753	1262	1229	14498	12783	14659	14447	-
AVE.	5.603	5.034	8.562	7.143	5.574	**2.503**	*3.795*	5.665	-
TRIMAVE.	4.841	4.273	7.500	5.583	5.072	**2.227**	*2.804*	4.400	-
STD.	6.619	5.662	9.411	8.812	6.221	**2.427**	*6.061*	9.224	-
Impro (AVE).	-	10.166	-52.797	-27.470	0.526	**55.326**	*32.267*	-1.091	-
Impro (TRIMAVE).	-	11.738	-54.921	-15.337	-4.767	**54.000**	*42.075*	9.101	-

TABLE IV: Performance comparison between GCNs.

Metrics \ GCNs	AIG		Skeleton(K=4)	
	HuberLoss	TrainTime	HuberLoss	TrainTime
GIN[33]	**0.252**	15829	**0.190**	12783
SAGE[34]	0.292	7996	0.325	**7312**
HOGA[12]	0.430	183353	0.432	62041
BoolGebra[14]	0.422	**6748**	0.253	13723

employed to assess the "arrival time" of the netlist generated for each design across its respective optimization sequence.

GCN-Based Model. Fig. 19 illustrates the GCN-based timing prediction model tailored for logic optimization. Training data is generated per the dataset flow in Fig. 13, with each sample comprising a Boolean dependency graph and its associated logic optimization sequence. The model aims to predict the timing quality of a Boolean network under a given optimization action. It consists of three core components: (1) a graph embedding module, employing a two-layer GCN for node feature aggregation followed by a readout layer (mean + max) to produce a graph-level feature vector; (2) a synthesis embedding module, encoding the discrete optimization sequence using four convolutional filters with kernel sizes {1×14, 1×15, 1×16, 1×17}; and (3) a timing quality prediction module, utilizing fully connected layers and a softmax activation to estimate the timing quality as a probability distribution across normalized timing range. The training process spans 500 epochs, with a learning rate of 0.001, and employs Huber-loss [38] as the loss function.

Evaluation. Table IV shows performance comparison results of various GCNs applied to two graph types: the original AIG and the skeleton graph by the proposed *BoolSkeleton* method with $K = 4$. It suggests that the GIN-based timing prediction model surpasses all the other models in the loss evaluation, although the training time is comparatively higher.

Table III reports the Mean Absolute Percentage Error (MAPE) for timing prediction across various graph-based methods applied to the dataset in Table II, with lower MAPE values signifying higher accuracy. Among the evaluated approaches, *BoolSkeleton* with $K = 4$ consistently outperforms others, achieving the lowest average MAPE (AVE: 2.503) and trimmed mean (TRIMAVE[4]: 2.227), surpassing the baseline AIG by 55.3% and 54.0%, respectively. In contrast, methods like *Alberaic Distance* (AVE: 8.562) and *Kron reduction* (AVE: 7.143) exhibit higher errors, underperforming AIG by 52.797% and 27.470%, respectively. The standard deviation (STD) across designs further underscores the stability of *BoolSkeleton* ($K = 4$: 2.427, lowest), while per-design STD (rightmost column) reveals circuit-specific variability, with designs like *ttt2* (14.070) showing greater sensitivity to skeletonization than *sin* (0.721). These results suggest that *BoolSkeleton* effectively balances accuracy and robustness, offering a superior approach for timing prediction.

VI. Discussion

In this section, we will discuss the advantages and limitations of the proposed *BoolSkeleton*.

[4]TRIMAVE is the average excluding the maximum and minimum values.

Advantages. *BoolSkeleton* is designed from the functionality viewpoint, enabling it to distill a consistent structural representation across diverse Boolean network variants. This method significantly enhances performance in functionality-related tasks of the Boolean networks by preserving essential Boolean dependency properties within structural variability. Additionally, the node-level fanin-limited homogeneous pattern reduction process efficiently extracts a coarse-grained skeleton from the Boolean network, facilitating the capture of high-dimensional representations. The parameter K of the pattern reduction operator can also control the coarsening ratio of the Boolean dependency graph. Experimental results validate these advantages, showcasing superior outcomes across multiple downstream applications. Furthermore, it holds the potential to improve functionality- and profile-related tasks, for example, it can help the matches between ports in Boolean matching [39], and it can enhance the optimization space exploration by PPA prediction [40], [41], [42], etc.

Limitations There is a reduction ratio for the graph coarsening method, and the parameter K of the pattern reduction operator is used to control the reduction ratio in $BoolSkeleton$. While this flexibility of K also expands the scope of analytical tasks, determining an optimal K value for specific Boolean networks and tasks poses a challenge. Consequently, an adaptive $BoolSkeleton$ tailored to individual tasks is also essential. For large-scale circuits, a potential solution is to combine with "BoolSkeleton" through graph partitioning algorithms, which is also consistent with the nature of large circuits that are generally composed of sub-modules.

VII. Conclusion

In this work, we proposed *BoolSkeleton* to address the challenge of the coarse-grained Boolean network representation in logic synthesis. *BoolSkeleton* is tailored to balance their static functionality and dynamic structural variability. Through an in-depth analysis, we identified critical attributes: Boolean dependency, reachability, reconvergence, and the conflict between static functionality and dynamic DAG. Leveraging these insights, *BoolSkeleton* employs a preprocessing phase to transform Boolean networks into dependency graphs, followed by an iterative node-level homogeneous pattern reduction process. This approach preserves coarse-grained functionality-related information while simplifying fine-grained topological structures, addressing the limitations of existing GNN-based methods that overly rely on local node embeddings. The experimental results of four downstream tasks demonstrate the efficacy of *BoolSkeleton*. Future work will focus on the integration with advanced machine learning platforms or frameworks to *BoolSkeleton*'s applicability across a broader range of Boolean network representation tasks.

References

[1] R. Brayton, G. Hachtel, and A. Sangiovanni-Vincentelli, "Multilevel logic synthesis," *Proceedings of the IEEE*, vol. 78, no. 2, pp. 264–300, 1990.

[2] G. D. Micheli, *Synthesis and Optimization of Digital Circuits*, 1st ed. McGraw-Hill Higher Education, 1994.

[3] A. Mishchenko and R. Brayton, "Simplification of non-deterministic multi-valued networks," in *Proceedings of the 2002 IEEE/ACM ICCAD*. New York, NY, USA: Association for Computing Machinery, 2002, p. 557–562. [Online]. Available: https://doi.org/10.1145/774572.774654

[4] A. Mishchenko, S. Chatterjee, and R. Brayton, "DAG-aware AIG rewriting: a fresh look at combinational logic synthesis," in *ACM/IEEE DAC*, 2006, pp. 532–535.

[5] B. R. Alan Mishchenko, "Scalable logic synthesis using a simple circuit structure," in *Proc. IWLS*, vol. 6, 2006, pp. 15–22.

[6] L. Ni, X. Li, B. Xie, and H. Li, "Boolean-aware boolean circuit classification: A comprehensive study on graph neural network," 2024. [Online]. Available: https://arxiv.org/abs/2411.10481

[7] J. Zhang, L. Ni, and et.al, "Enhanced fast boolean matching based on sensitivity signatures pruning," in *2021 IEEE/ACM ICCAD*, 2021, pp. 1–9.

[8] H. Cai, V. W. Zheng, and K. C.-C. Chang, "A comprehensive survey of graph embedding: Problems, techniques, and applications," *IEEE TKDE*, vol. 30, no. 9, pp. 1616–1637, 2018.

[9] M. Li, S. Khan, and et.al, "Deepgate: learning neural representations of logic gates," in *Proceedings of the 59th ACM/IEEE DAC*, ser. DAC '22. New York, NY, USA: Association for Computing Machinery, 2022, p. 667–672. [Online]. Available: https://doi.org/10.1145/3489517.3530497

[10] Z. Shi, H. Pan, and et.al, "Deepgate2: Functionality-aware circuit representation learning," in *2023 IEEE/ACM ICCAD*, 2023, pp. 1–9.

[11] Z. Shi, Z. Zheng, S. Khan, J. Zhong, M. Li, and Q. Xu, "Deepgate3: Towards scalable circuit representation learning," *CoRR*, vol. abs/2407.11095, 2024. [Online]. Available: https://doi.org/10.48550/arXiv.2407.11095

[12] C. Deng, Z. Yue, and et.al, "Less is more: Hop-wise graph attention for scalable and generalizable learning on circuits," in *Proceedings of the 61st ACM/IEEE DAC*, ser. DAC '24. New York, NY, USA: Association for Computing Machinery, 2024. [Online]. Available: https://doi.org/10.1145/3649329.3657386

[13] J. Liu, J. Zhai, M. Zhao, Z. Lin, B. Yu, and C. Shi, "Polargate: Breaking the functionality representation bottleneck of and-inverter graph neural network," in *2024 IEEE/ACM ICCAD*. [Online]. Available: https://www.cse.cuhk.edu.hk/~byu/papers/C233-ICCAD2024-PolarGate-slides.pdf

[14] Y. Li, A. Agnesina, Y. Zhang, H. Ren, and C. Yu, "Boolgebra: Attributed graph-learning for boolean algebraic manipulation," in *2024 DATE*, 2024, pp. 1–2.

[15] A. Loukas, "Graph reduction with spectral and cut guarantees," *Journal of Machine Learning Research*, vol. 20, no. 116, pp. 1–42, 2019. [Online]. Available: http://jmlr.org/papers/v20/18-680.html

[16] A. Loukas and P. Vandergheynst, "Spectrally approximating large graphs with smaller graphs," in *Proceedings of the 35th International Conference on Machine Learning*, ser. Proceedings of Machine Learning Research, J. Dy and A. Krause, Eds., vol. 80. PMLR, 10–15 Jul 2018, pp. 3237–3246. [Online]. Available: https://proceedings.mlr.press/v80/loukas18a.html

[17] D. Ron, I. Safro, and A. Brandt, "Relaxation-based coarsening and multiscale graph organization," *Multiscale Modeling & Simulation*, vol. 9, no. 1, pp. 407–423, 2011. [Online]. Available: https://doi.org/10.1137/100791142

[18] O. E. Livne and A. Brandt, "Lean algebraic multigrid (lamg): Fast graph laplacian linear solver," *SIAM Journal on Scientific Computing*, vol. 34, no. 4, pp. B499–B522, 2012. [Online]. Available: https://doi.org/10.1137/110843563

[19] F. Dorfler and F. Bullo, "Kron reduction of graphs with applications to electrical networks," *IEEE Transactions on Circuits and Systems I: Regular Papers*, vol. 60, no. 1, pp. 150–163, 2013.

[20] W. Schilp and P. Maurer, "Unit delay simulation with the inversion algorithm," in *Proceedings of ICCAD*, 1996, pp. 412–417.

[21] P. Fišer, J. Schmidt, and J. Balcárek, "Sources of bias in eda tools and its influence," in *17th International Symposium on Design and Diagnostics of Electronic Circuits & Systems*, 2014, pp. 258–261.

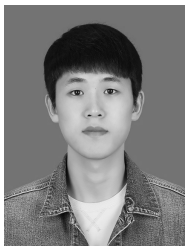
[22] M. Hashemi, S. Gong, and et.al, "A comprehensive survey on graph reduction: sparsification, coarsening, and condensation," in *Proceedings of the 33rd IJCAI*, 2024. [Online]. Available: https://doi.org/10.24963/ijcai.2024/891

[23] C. Liu, Y. Zhan, and et.al, "Graph pooling for graph neural networks: Progress, challenges, and opportunities," in *Proceedings of the 32nd IJCAI-23*, E. Elkind, Ed. IJCAI, 8 2023, pp. 6712–6722, survey Track. [Online]. Available: https://doi.org/10.24963/ijcai.2023/752

[24] T. Nkgau and G. Anderson, "Graph similarity algorithm evaluation," in *2017 Computing Conference*, 2017, pp. 272–278.

[25] C. Coupette and J. Vreeken, "Graph similarity description: How are these graphs similar?" in *Proceedings of the 27th ACM SIGKDD*

Conference on Knowledge Discovery & Data Mining. ACM, Aug. 2021, p. 185–195. [Online]. Available: http://dx.doi.org/10.1145/3447548.3467257
[26] C. Albrecht, "Iwls 2005 benchmarks," in *IEEE IWLS*, 2005.
[27] L. Amarú, P.-E. Gaillardon, and G. De Micheli, "The epfl combinational benchmark suite," in *IEEE IWLS*, 2015.
[28] M. Defferrard, L. Martin, R. Pena, and N. Perraudin, "Pygsp: Graph signal processing in python." [Online]. Available: https://github.com/epfl-lts2/pygsp/
[29] R. K. Brayton and A. Mishchenko, "ABC: A System for Sequential Synthesis and Verification," https://github.com/berkeley-abc/abc, accessed: 2024-02-01.
[30] L. Ni, R. Wang, and et.al, "Openls-dgf: An adaptive open-source dataset generation framework for machine learning tasks in logic synthesis," *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, pp. 1–1, 2025.
[31] M. Fey and J. E. Lenssen, "Fast graph representation learning with pytorch geometric," 2019. [Online]. Available: https://arxiv.org/abs/1903.02428
[32] X. Li, S. Tao, and et al, "iEDA: An Open-Source Intelligent Physical Implementation Toolkit and Library," 2023. [Online]. Available: https://arxiv.org/abs/2308.01857
[33] K. Xu, W. Hu, J. Leskovec, and S. Jegelka, "How powerful are graph neural networks?" in *International Conference on Learning Representations*, 2019. [Online]. Available: https://openreview.net/forum?id=ryGs6iA5Km
[34] W. L. Hamilton, R. Ying, and J. Leskovec, "Inductive representation learning on large graphs," in *Proceedings of the 31st International Conference on NeurIPS*, ser. NIPS'17. Red Hook, NY, USA: Curran Associates Inc., 2017, p. 1025–1035.
[35] A. B. Chowdhury, B. Tan, R. Karri, and S. Garg, "Openabc-d: A large-scale dataset for machine learning guided integrated circuit synthesis," 2021.
[36] N. Wu, J. Lee, Y. Xie, and C. Hao, "Lostin: Logic optimization via spatio-temporal information with hybrid graph models," in *2022 IEEE 33rd International Conference on Application-specific Systems, Architectures and Processors (ASAP)*, 2022, pp. 11–18.
[37] F. Faez, R. Karimi, and et.al, *MTLSO: A Multi-Task Learning Approach for Logic Synthesis Optimization*. New York, NY, USA: Association for Computing Machinery, 2025, p. 72–78. [Online]. Available: https://doi.org/10.1145/3658617.3697721
[38] P. J. Huber, "Robust Estimation of a Location Parameter," *The Annals of Mathematical Statistics*, vol. 35, no. 1, pp. 73 – 101, 1964. [Online]. Available: https://doi.org/10.1214/aoms/1177703732
[39] H. Katebi and I. L. Markov, "Large-scale boolean matching," in *2010 DATE*, 2010, pp. 771–776.
[40] P. Sengupta, A. Tyagi, Y. Chen, and J. Hu, "How good is your verilog rtl code? a quick answer from machine learning," in *2022 IEEE/ACM ICCAD*, 2022, pp. 1–9.
[41] D. S. Lopera and W. Ecker, "Applying gnns to timing estimation at rtl : (invited paper)," in *2022 IEEE/ACM ICCAD*, 2022, pp. 1–8.
[42] W. Fang, Y. Lu, and et.al, "Masterrtl: A pre-synthesis ppa estimation framework for any rtl design," in *2023 IEEE/ACM ICCAD*, 2023, pp. 1–9.



Liwei Ni is currently pursuing the Ph.D. degree with the State Key Laboratory of Processors, Institute of Computing Technology, Chinese Academy of Sciences, Beijing, China, and also with Pengcheng Laboratory. His current research interests include logic optimization, technology mapping, and formal verification.

Jiaxi Zhang received his Ph.D. degree from the School of Computer Science, Peking University, Beijing, China, in 2022. He is currently a Postdoctoral Researcher at the School of Computer Science, Peking University. His research interests include electronic design automation, hardware security, and heterogeneous computing.

Shenggen Zheng Zheng received his Ph.D. in Computer Science from Sun Yat-sen University in 2012. From 2012 to 2015, he was a postdoctoral researcher under the mentorship of Jozef Gruska and a visiting researcher in Andris Ambainis' group. He then joined Sun Yat-sen University as a Research Scientist from 2015 to 2018. From 2018 to 2024, he was affiliated with Pengcheng Laboratory and the Institute for Quantum Science and Engineering at the Southern University of Science and Technology. Currently, he is a Researcher at the Quantum Science Center of the Guangdong-Hong Kong-Macao Greater Bay Area. His research interests include quantum computing, quantum query complexity, quantum machine learning, and EDA.

Junfeng Liu received the PhD degree in computer science from Beihang University, China, in 2024. He is currently a postdoctoral researcher at Pengcheng Laboratory, China. His research interests include EDA logic synthesis and graph data management.

Xingyu Meng received the PhD degree from the department of Electrical and Computer Engineering at the University of Texas at Dallas as part of the Trustworthy and Intelligent Embedded System (TIES) lab. His research interests include hardware and system security, Trojan detection, and hardware verification. His research has been published in Design Automation Conference (DAC), IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems (TCAD), etc.

Biwei Xie is an Associate Professor at the Institute of Computing Technology, Chinese Academy of Sciences. He received his Ph.D. degree from the same institution in 2018. His research interests encompass open EDA, open-source chip design, high-performance computing, and computer architecture. His work has been published in leading international conferences such as CGO, ICS, ICCAD, and DATE.

Xingquan Li received the Ph.D degree from Fuzhou University in 2018. He is an Associate Professor at Pengcheng Laboratory. His research interests include EDA and AI for EDA. His team has developed an open-source infrastructure of EDA and toolchain (iEDA). He has published over 60 papers, and received three First-place Awards from ICCAD@CAD Contest in 2017, 2018, and 2022. In 2020, he received the Application Award of Operations Research from the Operations Research Society of China, and the Best Paper Award from ISEDA 2023.

Huawei Li (M'00–SM'09) received the B.S. degree in computer science from Xiangtan University, Xiangtan, China, in 1996, and the M.S. and Ph.D. degrees from the Institute of Computing Technology (ICT), Chinese Academy of Sciences (CAS), Beijing, China, in 1999 and 2001, respectively. She has been a Professor with ICT and CAS since 2008. Her current research interests include testing of VLSI/SoC circuits, electronic design automation and computer architecture. She has published more than 300 technical papers and holds 35 Chinese patents in the above areas. She currently serves as the Chair of the China Computer Federation (CCF) Technical Committee on Integrated Circuit Design, and the Associate Editor of IEEE Transactions on Very Large Scale Integration (VLSI) Systems, IEEE Design & Test, and Journal of Computer Science & Technology. She has also served on the technical program committees for several IEEE/ACM conferences.