CHOP: Clustered Hybrid Optimization for Logic Synthesis with Self-Supervised Prediction

Rongliang Fu, Ran Zhang, Ziyang Zheng, Zhengyuan Shi, Yuan Pu, Junying Huang, Bei Yu, Qiang Xu, and Tsung-Yi Ho *Fellow, IEEE*

***Abstract*—Hybrid optimization is emerging as a promising approach for enhancing logic synthesis by combining multiple logic optimization methods. This paper introduces CHOP, a novel framework that leverages vertex-level feature extraction and optimization-aware clustering to achieve significant improvements in the efficiency and quality of logic synthesis. Unlike existing cluster-level approaches that follow a "partition-then-predict" paradigm, CHOP operates under a "predict-then-partition" approach where the optimization method most appropriate for a given circuit vertex is determined by analyzing its local structural and functional context. To capitalize on this insight, CHOP extracts a subgraph centered on each vertex and calculates its circuit cost, which serves as a vertex feature for the subsequent circuit partitioning process. Furthermore, we introduce a self-supervised prediction model to obtain these vertex features efficiently. Following circuit partitioning, CHOP determines the optimization method for each partition and then iteratively applies these methods to derive the optimized circuit. The experimental results from logic optimization and LUT mapping tasks on the EPFL benchmark demonstrate the effectiveness and efficiency of CHOP.**

***Index Terms*—Hybrid logic optimization, circuit partitioning, self-supervision model, logic synthesis.**

I. Introduction

Logic synthesis plays a crucial role in modern electronic design automation (EDA) [1], involving logic optimization and technology mapping. Early efforts focused on simplifying Boolean expressions using algebraic techniques. As the complexity of digital circuits increased, the need for more advanced optimization techniques led to the development of multi-level logic optimization [2]. This approach considers entire networks of logic gates, rather than isolated expressions, thereby enhancing optimization efficacy. The introduction of directed acyclic graphs (DAGs) to represent Boolean networks enabled more effective logic optimization by facilitating the exploration of different structural configurations. Various types of graph structures, such as AND-Inverter graphs (AIGs) [3], Majority-Inverter graphs (MIGs) [4], XOR-AND-Inverter graphs (XAGs) [5], and XOR-Majority-Inverter graphs (XMGs) [6], have been developed to effectively capture different logic functions and facilitate algebraic manipulation. For example, Fig. 1 shows the different graph representations for the 1-bit full adder. The AIG representation requires 8 nodes and has a depth of 5, while the MIG representation requires only 3 nodes and has a depth of 2. The XAG representation requires 5 nodes and has a depth of 3, while the XMG representation requires only 3 nodes and has a depth of 2. This comparison underscores the diverse capabilities of different graph structures in optimizing logic functions.

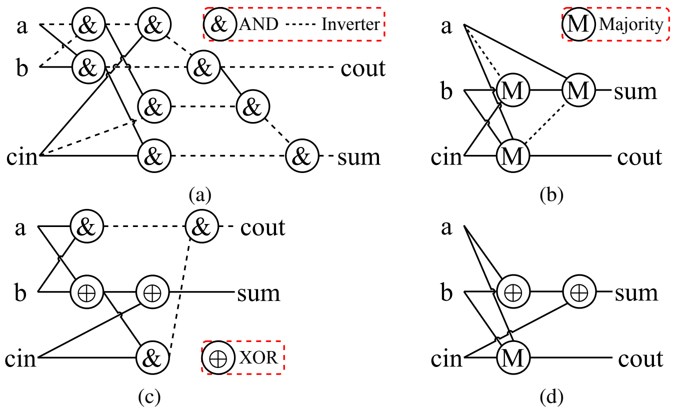


Fig. 1. Graph representations for the 1-bit full adder using (a) AIG with 8 nodes and a depth of 5, (b) MIG with 3 nodes and a depth of 2, (c) XAG with 5 nodes and a depth of 3, and (d) XMG with 3 nodes and a depth of 2.

The introduction of these graphs has facilitated the development of numerous logic optimization techniques, such as rewriting [5], [7]–[9], resynthesis [10], [11], and resubstitution [12]–[14]. Rewriting involves optimizing the number of gates or depth of logic circuits by rearranging or substituting specific Boolean networks within the circuit with their equivalent forms. Resynthesis focuses on the comprehensive reconstruction or reorganization of the original subcircuit into a new logic structure. In contrast, resubstitution involves replacing redundant nodes with existing ones, thereby avoiding the introduction of new nodes. However, these methods mainly focus on a single type of graph representation.

Due to the inherent complexity of graph structures, hybrid optimization, a combination of these circuit graph optimization techniques, has the potential to significantly enhance circuit performance [15]–[18], as shown in Fig. 2. Luca et al. pro-

The research work described in this paper was conducted in the JC STEM Lab of Intelligent Design Automation funded by The Hong Kong Jockey Club Charities Trust and was supported in part by the Research Grants Council of Hong Kong SAR (Grant No. CUHK14207523), and in part by the National Natural Science Foundation of China (Grants No. 62302477).

Rongliang Fu, Ziyang Zheng, Zhengyuan Shi, Yuan Pu, Bei Yu, Qiang Xu and Tsung-Yi Ho are with the Department of Computer Science and Engineering, The Chinese University of Hong Kong, Hong Kong 999077, China. E-mail: {rlfu, zyzheng23, zyshi, ypu, byu, qxu, tyho}@cse.cuhk.edu.hk.

Ran Zhang and Junying Huang are with the SKLP, Institute of Computing Technology, CAS, Beijing 100190, China. E-mail: {zhangran23s, huangjunying}@ict.ac.cn.

Corresponding author: huangjunying@ict.ac.cn.

TABLE I. Comparison of CHOP and existing hybrid logic optimization frameworks.

Aspect	LSOracle [16]	HeLO [17]	CHOP
Prediction Level	Cluster-level	Cluster-level	**Vertex-level**
Prediction Phase	After clustering	After clustering	**Before clustering**
Clustering Method	K-way Hypergraph	Agglomerative	**Hierarchical Clustering**

posed MixSyn [15], which combines different logic optimization techniques to improve circuit performance. MixSyn first performs XOR optimization for a given Boolean network, and then directly splits the optimized Boolean network into two parts, composed of (i) XOR nodes and (ii) remaining nodes, respectively. Finally, it performs AND/OR optimization on the second part, and then merges the resulting network and the first part to obtain the final optimized Boolean network. Thus, MixSyn merely combines XOR and AND/OR optimization techniques, without considering the unique characteristics of the circuit design. Neto et al. proposed LSOracle [16], which identifies clusters within a circuit and assigns a specific graph optimization technique to each cluster to explore a better circuit design. Although LSOracle achieves promising improvements in the area-delay product (ADP), it has notable shortcomings:

- It relies on the general graph partition tool, KaHyPar [19], to partition the circuit, without accounting for the unique characteristics of circuit design.
- It represents Boolean functions using Karnaugh-map images and employs a traditional neural network model to predict the optimization method of each cluster, which may limit prediction performance on complex circuits.

Besides, Pu et al. proposed HeLO [17], a heterogeneous DAG-based logic optimization framework that combines graph learning and hierarchical clustering. HeLO clusters logic components based on structural and functional similarity and uses a graph neural network (GNN) model to generate topological-functional embeddings for predicting the best-fit DAG type for each subcircuit. However, existing hybrid optimization methods face several fundamental limitations that constrain their effectiveness. First, cluster-level predictions lack the fine-grained vertex-level feature extraction necessary for optimal Boolean network partitioning. Second, the sequential "partition-then-predict" paradigm prevents feedback from prediction results to improve clustering quality. Finally, these methods often rely on general-purpose clustering techniques that fail to capture the intricate relationship between circuit structures and their optimal optimization strategies, leading to suboptimal partitioning decisions.

To address these challenges, this paper proposes CHOP, a hybrid optimization framework that incorporates circuit-specific characteristics into the optimization process. As shown in TABLE I, CHOP differs from traditional logic optimization frameworks. Unlike previous methods that rely on cluster-level prediction, CHOP extracts structural and functional features from each vertex in a Boolean network. These features guide the network partitioning and enable the selection of optimal optimization strategies for each cluster, thereby effectively managing the complexity of modern circuit structures. Overall,

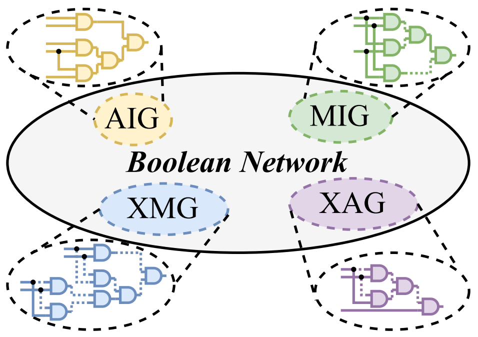


Fig. 2. A hybrid optimization example.

this paper makes the following contributions.

- **Paradigm shift**: A fundamental shift from "partition-then-predict" to "predict-then-partition" paradigm, enabling optimization-aware circuit clustering that naturally aligns with synthesis objectives.
- **Vertex-level prediction**: A novel vertex-level feature extraction method using self-supervised learning that captures fine-grained optimization preferences, surpassing traditional cluster-level approaches.
- **Hierarchical clustering**: A circuit-specific hierarchical partitioning algorithm that integrates both structural topology and functional optimization tendencies through adaptive masking strategies.
- **Optimization selection**: An intelligent optimization method selection strategy that leverages predicted vertex features to determine the most suitable synthesis technique for each cluster.

Moreover, experimental results on the EPFL benchmark [20] demonstrate the effectiveness and efficiency of our proposed framework compared with the state-of-the-art.

- For logic optimization, CHOP achieves average node-depth-product (NDP) reductions of 16.44%, and 13.33%, with average speeds of 1.92x and 2.98x, compared with the state-of-the-art hybrid optimization methods.
- For 6-LUT mapping, CHOP achieves average ADP reductions of 17.99% and 36.42%, with average speeds of 1.78x and 2.48x, compared with the state-of-the-art hybrid optimization methods.

The rest of this paper is organized as follows: Section II provides some background information. Section III introduces the terminology used and our problem formulation. Section IV introduces our proposed hybrid optimization framework. Section V presents the experimental methods and results. Finally, Section VI summarizes our work.

II. Preliminaries

A. Boolean Network

A Boolean network [21] is a mathematical model that characterizes a system of interconnected binary variables, where each variable can assume one of two possible values: true (1) or false (0). In a Boolean network, the state of each vertex at time $t + 1$ is determined by a Boolean function that depends on the states of a subset of the other vertices at time

t. Thus, the dynamics of the network are governed by the following equation:

$$x_i(t+1) = f_i(x_{i_1}(t), x_{i_2}(t), ..., x_{i_k}(t)), \quad (1)$$

where $x_i(t)$ is the state of vertex i at time t, f_i is the Boolean function associated with vertex i, and $x_{i_1}(t), x_{i_2}(t), ..., x_{i_k}(t)$ are the states of the k input vertices that regulate vertex i. The simplicity of Boolean networks, coupled with their capacity to model complex interactions, makes them a powerful tool for both theoretical analysis and practical applications in diverse domains. Notably, in digital circuit design, Boolean networks facilitate the representation of both combinational and sequential circuits, thereby enhancing our understanding and manipulation of these systems.

B. Logic Representation

In logic synthesis, Boolean networks are typically modeled using DAGs, which provide multiple representation structures. These representations provide structured and efficient ways for describing and optimizing Boolean networks, facilitating the analysis and transformation of Boolean functions. The following four structures are prevalent in representing Boolean networks, each with distinct characteristics and applications.

AIG [3] is a widely used representation in which the vertices correspond to AND operations, and the edges represent signals or their inversions.

MIG [4] incorporates majority operations (a 3-input majority is $\mathrm{M}(a,b,c) = (a \wedge b) \vee (a \wedge c) \vee (b \wedge c)$) alongside inverters, allowing for a more compact and expressive representation of certain logic functions.

XAG [5] merges XOR operations with AND operations and inverters, particularly advantageous for applications involving parity checks and error detection, where XOR operations play a crucial role.

XMG [6] integrates XOR operations, majority operations, and inverters, suitable for a wide range of applications, including quantum circuit synthesis and cryptographic function implementations.

Overall, as stated in Theorem 1, the XMG has the most expressive power, as it can represent all other structures. The AIG is the simplest structure, while the MIG and XAG are intermediate representations that can capture specific logic functions more efficiently than the AIG.

Theorem 1. *AIG $\subset$ MIG / XAG $\subset$ XMG.*

Proof. In all graphs, inverters are represented by complemented edge markers.

AIG $\subset$ XAG Since AIG nodes only have AND operations, an AIG is trivially an XAG where all XOR nodes are unused, *i.e.*, AIG $\subset$ XAG.

AIG $\subset$ MIG / XMG Since an AND operation $f = a \wedge b$ is equivalent to a 3-input majority operation with one input tied to 0: $f = \mathrm{M}(a,b,0)$, an AIG node is always a special case of a majority node, *i.e.*, AIG $\subset$ MIG / XMG.

MIG $\subset$ XMG Since MIG nodes only have majority operations, an MIG is trivially an XMG where all XOR nodes are unused, *i.e.*, MIG $\subset$ XMG.

XAG $\subset$ XMG Since XAG nodes have XOR and AND operations and an AND node is always a special case of a majority node, an XAG is always a special case of an XMG, *i.e.*, XAG $\subset$ XMG.

Therefore, we conclude AIG $\subset$ MIG / XAG $\subset$ XMG. □

Algorithm 1: Leiden algorithm.

```
  Input: A network $G(V,E)$.
  Output: Partition of vertices into communities $cs$.
1 $cs \leftarrow \{\{v\} \mid v \in V\}$
2 while modularity $Q$ improves do
3     for $v \in V$ do
4         $cs \leftarrow$ Move $v$ for maximizing $Q$
5     $cs \leftarrow$ Identify potential subcommunities in $cs$
6     $G(V,E) \leftarrow$ Each $c \in cs$ is regarded as a vertex
7 return $cs$
```

C. Community Detection

In complex networks, nodes tend to cluster into relatively dense groups, commonly referred to as communities, whose modular structure is typically unknown in advance. Community detection [22] thus becomes a crucial problem for understanding the structure of large and complex networks. Among the best-known methods for community detection are those based on optimizing modularity [23], a measure of partition quality, such as the Louvain [24] and Leiden [25] algorithms. These methods identify communities by maximizing modularity Q, effectively performing network partitioning tasks. Although the definitions of Q vary slightly, the primary formula is as follows:

$$Q = \frac{1}{2m}\sum_{u,v}(A_{u,v} - \frac{k_u k_v}{2m})\delta(c_u, c_v), \quad (2)$$

where m is the total number of edges in the network, k_v is the degree of vertex v, A is the adjacency matrix of the network, and $\delta(c_u, c_v)$ is an indicator function that equals 1 if vertices u and v belong to the same community, and 0 otherwise.

The Louvain algorithm is characterized by its simplicity and elegance, optimizing the modularity through two fundamental phases: (i) local movement of nodes; and (ii) aggregation of the network. In the local movement phase, individual nodes are relocated to the community that yields the largest increase in quality function. Subsequently, in the aggregation phase, an aggregate network is constructed based on the partition obtained in the local movement phase, and each community in this partition is treated as a node in the aggregate network. The two phases are repeated until there is no further increase in the quality function. However, the Louvain algorithm may yield arbitrarily poorly connected communities, even internally disconnected communities in the worst case [25].

To address this problem, the Leiden algorithm [25] introduces a combination of smart local movement, fast local movement, and random neighbor movement to produce partitions in which all communities are well-connected. Also, the Leiden algorithm is faster than the Louvain algorithm and can find superior partitions. Algorithm 1 outlines the process of the Leiden algorithm, which optimizes modularity Q by iteratively refining community partitions. The algorithm begins with an initial partition where each vertex belongs to its own community (line 1). It then iteratively performs the following steps:

Local moving phase: Vertices are reassigned to neighboring communities to maximize modularity gain (lines 3-4).

Refinement phase: The resulting communities are divided into smaller and well-connected subcommunities to ensure connectivity (line 5).

Aggregation phase: The network is coarsened by treating each refined community as a single vertex (line 6).

This process repeats until convergence (*i.e.*, no further improvement in modularity), producing a high-quality partition with connectivity guarantees (lines 2-6).

During hybrid logic optimization, it is often difficult to identify which vertices should be clustered together into a subcircuit to yield optimal optimization results. Consequently, this paper will integrate community detection methods with logic synthesis to enhance circuit clustering. This approach aims to overcome the complexities inherent in modern circuit designs and facilitate the exploration of more effective circuit design strategies.

D. Representation Learning for Circuit

Circuit representation learning has emerged as a pivotal area in EDA, reflecting the broader trend in artificial intelligence toward developing general representations that can be applied to a variety of downstream tasks. Within this field, the DeepGate family of models [26], [27] has pioneered the use of GNNs to encode AIGs, enabling a wide range of EDA applications, such as testability analysis [28], power estimation [29], and SAT solving [30], [31]. Notable advancements include the introduction of Gamora [32] and HOGA [33], which enhance reasoning capabilities by effectively representing both logic gates and their associated cones. Additionally, PolarGate [34] addresses the challenges related to functionality representation by leveraging the principles of ambipolar states. Despite the progress made with GNNs, challenges such as over-squashing and over-smoothing persist. To mitigate these issues, the recent work of DeepGate3 [35] utilizes DeepGate2 as a tokenizer and then leverages the global aggregation mechanism of transformers with a connective mask to enhance circuit representation. As a result, DeepGate3 demonstrates enhanced performance across various pre-training tasks at both gate-level and graph-level.

III. Problem Formulation

A. Terminology

A DAG serves as an effective representation of a Boolean network, denoted as $G(V, E)$. In this representation, the set of vertex V typically includes a collection of primary inputs (PIs), primary outputs (POs), and internal nodes within the network. The directed edge set $E \subseteq V \times V$ illustrates the connectivity relationships among the vertices, with the orientation of these edges indicating the direction of the data flow within the circuit. For any vertex $v \in V$, we define FI(v) and FO(v) to represent the sets of fan-in and fan-out vertices, respectively. Consequently, for any vertex $u \in \text{FI}(v)$ or $v \in \text{FO}(u)$, there exists a directed edge $(u, v) \in E$. In particular, for any $i \in$ PIs or $o \in$ POs, their respective sets of fan-in and fan-out vertices are empty. Furthermore, the depth of a Boolean network is determined as the maximum logic level among all POs, and the size of the network is the total number of vertices within G. For any vertex v, the logic level ι of a vertex v is recursively defined as $\iota(v) = \max_{u \in \text{FI}(v)} \iota(u) + 1$.

B. Problem Formulation

Hybrid logic optimization involves the application of various circuit graphs and the corresponding logic optimization methods to a Boolean network. Since the circuit design often involves multiple metrics, such as delay and area, the optimization process seeks to achieve a balance among these metrics. This paper specifically focuses on optimizing the area-delay product (ADP), a widely used metric in circuit design that effectively balances both delay and area. Hence, the optimization problem in this paper can be formulated as follows:

Input: A Boolean network $G(V, E)$.

Output: An optimized circuit $G'(V', E')$.

Constraints: The logic functions remain the same before and after optimization.

Goal:

$$\min \{\text{ADP} = \mathit{delay} \times \mathit{area}\}, \tag{3}$$

where minimizing the ADP facilitates a more effective balance between delay and area, thereby optimizing the overall performance of the circuit.

This problem poses several substantial challenges. Firstly, it is essential to quantify the closeness between vertices within a Boolean network. Although the relationships among vertices may not be immediately apparent from the circuit's representation, they do objectively exist and can be exploited to enhance the partition of G. We define the closeness between vertices u and v as $\rho_{u,v}$:

$$\rho_{u,v} = \begin{cases} 1 & \text{if } (u, v) \in E, \\ 0 & \text{otherwise}, \end{cases} \tag{4}$$

where (u, v) denotes a directed edge from u to v. Secondly, the effective partition of the Boolean network is vital, as the quality of the partition directly influences the outcome of the hybrid optimization process. Lastly, executing all optimization methods on each cluster can be exceedingly time-consuming. If we can harness the properties of the cluster to directly identify which circuit representation and corresponding optimization method are most suitable for a given cluster, this predictive capability can substantially reduce the overall runtime of the optimization process. We will address these challenges within the framework of CHOP, thereby enhancing circuit design.

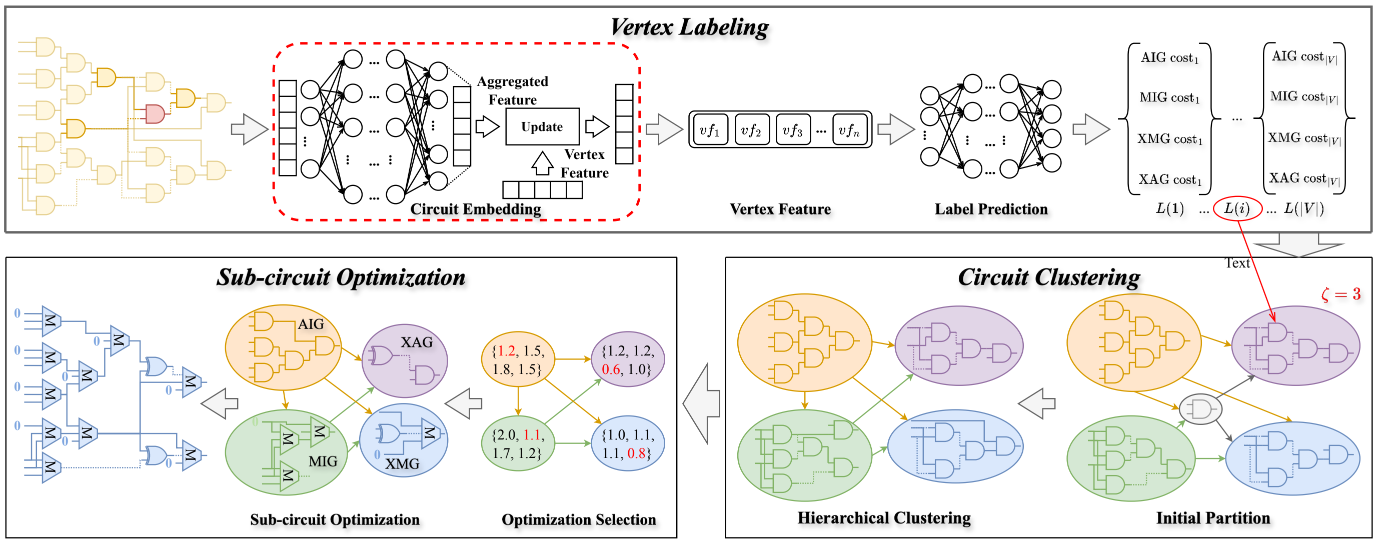


Fig. 3. The flow of the CHOP framework.

IV. CHOP

To address these limitations and incorporate circuit-specific characteristics into the hybrid optimization process, this section presents CHOP, a hybrid optimization framework that fundamentally shifts from cluster-level to vertex-level analysis. Unlike traditional methods that rely on post-partitioning prediction, CHOP extracts structural and functional features from each vertex in a Boolean network and uses these features to guide optimization-aware partitioning. This paradigm shift enables more precise clustering that naturally aligns with optimization objectives, thereby addressing the core limitations of existing approaches.

As illustrated in Fig. 3, CHOP operates through three main stages: (1) *vertex labeling*, where each vertex is assigned an optimization preference label based on its local structural and functional context; (2) *circuit clustering*, where vertices with similar labels are grouped using a hierarchical clustering algorithm; and (3) *sub-circuit optimization*, where each cluster is optimized using its most suitable optimization method, and the results are merged to form the final optimized circuit.

A. Vertex Labeling

Unlike general graphs, Boolean networks exhibit not only complex interconnect topologies but also rich functional semantics, as each vertex corresponds to a specific logic gate or sub-function. This dual complexity poses significant challenges for downstream optimization tasks, which often require detailed, fine-grained knowledge of circuit characteristics at the vertex level. To address these challenges, we propose a learning-based feature extraction method that efficiently encodes both the structural and functional attributes of each vertex within the network. In contrast to traditional approaches that rely on manual feature engineering, which are often time-consuming and difficult to generalize, our method leverages a pre-trained neural model to infer expressive representations of circuit elements automatically.

1) Feature Selection: Given that the optimization potential of a circuit vertex is fundamentally determined by its local structural context and functional dependencies, we utilize the features of a subgraph surrounding a vertex to represent its optimization characteristics. This approach is theoretically grounded in the principle that logic optimization techniques operate on localized circuit patterns, where the effectiveness of a specific optimization method depends heavily on the surrounding logic structure. In the context of a Boolean network, we introduce a partial order $\preccurlyeq_k$, defined such that for any two vertices u and v, the relation $u \preccurlyeq_k v$ holds if there exists a path from u to v with a length of at most k. For a specific vertex $v \in V$, we define its expanded subgraph with degree k as follows:

$$g_k(v) = \{u \in V : u \preccurlyeq_k v\} \cup \{u \in V : v \preccurlyeq_k u\}. \quad (5)$$

As an illustrative example, the red vertex located in the upper-left corner of Fig. 3 and its surrounding highlighted vertices collectively construct a subgraph with $k = 1$.

After obtaining the expanded subgraph $g_k(v)$ for vertex v, the next step involves identifying the optimal circuit graph for $g_k(v)$. The ADP values produced through different optimization methods for the expanded subgraph $g_k(v)$ may exhibit similarities. Therefore, relying solely on the circuit graph with minimum cost can not accurately reflect the characteristics of a vertex. This paper denotes the costs of different circuit graphs applied to the expanded subgraph as a vector: $Cost = [cost_1, cost_2, \cdots, cost_n]$, where $cost_i$ represents the ADP generated using the optimization method for the i^{th} structure of circuit graphs. To facilitate the analysis of different vertex features, we apply softmax normalization to generate the final label vector L for each vertex:

$$L = \text{softmax}(Cost). \quad (6)$$

The processed subgraph set $S(G)$, together with the label set L, serves as training data for learning the labels of vertices within the Boolean network. These labels subsequently guide both the partition and prediction tasks. In the following sections, we will introduce the graph neural network architecture adopted for label prediction.

2) Model Selection: To reduce the time-consuming process of acquiring vertex labels L, an accurate and efficient prediction model is essential to lower the overall computational cost. Our task requires considering both structural and functional aspects of logic synthesis, making models that focus solely on functionality, such as FGNN [36] and PolarGate [34], unsuitable. Additionally, the partitioned subcircuits in our dataset vary widely in size, demanding a model that generalizes well across different scales. Since incorporating model predictions adds time overhead, it is crucial to use a neural network with low computational complexity to keep the overall runtime reasonable. Given these criteria, we chose DeepGate, a state-of-the-art transformer-based model that captures both structural and functional features while offering strong generalization capabilities [35]. Note that our framework remains flexible and configurable, allowing alternative models to be integrated as needed.

3) Label Prediction: According to the subgraph expansion method in Section IV-A1, we obtain a set of subgraphs $S(G)$. For each subgraph $g_k(v)$, we prepare the corresponding label $L(v) = [l_1(v), l_2(v), \cdots, l_n(v)]$. Subsequently, we train the DeepGate model with these subgraphs and their corresponding labels, as illustrated in the upper middle part of Fig. 3. Given a vertex v, to predict its corresponding label $\hat{L}(v)$, we first get the expanded subgraph $g_k(v)$ and then use the DeepGate model to embed the subgraph:

$$\mathrm{VF} = \mathit{DeepGate}(g_k(v)). \tag{7}$$

$\mathrm{VF} = \{vf_1, vf_2, \cdots, vf_{|V|}\}$ represents the embedding sequence of the subgraph, where $|V|$ denotes the number of vertices $\{v_i, i = 1, 2, ..., |V|\}$ within $g_k(v)$, and vf_{v_i} represents the embedding sequence of vertex v_i. Then, we predict the $\hat{L}(v)$ by concatenating both functional and structural embedding and feeding the embedding into a Multi-Layer Perceptron (MLP):

$$\hat{L}(v) = \mathrm{MLP}\left(\mathrm{VF}\right). \tag{8}$$

Finally, we calculate the loss between the predicted label $\hat{L}(v)$ and the ground truth $L(v)$ after the softmax function:

$$loss = \|softmax(L(v)), softmax(\hat{L}(v))\|_1. \tag{9}$$

During the process of using DeepGate to predict the label set L for all vertices in the Boolean network G, each vertex v iteratively aggregates information from its neighbors and updates its own representation. This iterative message-passing mechanism is referred to as circuit embedding. The outcome of this embedding process is formally defined in Equation (7), and its structural location is highlighted by the red dotted region in the upper half of Fig. 3. The resulting vertex representations are subsequently fed into a multi-layer perceptron, as described in Equation (8), to generate the final label predictions. This prediction stage corresponds to the area shown in the upper right corner of Fig. 3.

Algorithm 2: Circuit Clustering.

```
   Input: A Boolean network $G(V, E)$, labels $L$, $\zeta$.
   Output: Partition results $cs$.
1  $\iota \leftarrow$ Calculate the logic levels of all vertices
2  $P \leftarrow$ Get POs in $G$ sorted by $\iota$
3  $cs \leftarrow \emptyset$
4  while $P \neq \emptyset$ do
5      $r \leftarrow \mathrm{argmax}_{r \in P} \iota(r)$, $P \leftarrow P \setminus \{r\}$
6      $c \leftarrow \emptyset$, $q \leftarrow \emptyset$
7      $v \leftarrow r$
8      while $\iota(r) - \iota(v) < \zeta$ do
9          if $\iota(r) - \iota(v) > \zeta/2 \wedge |c| < 2^{\zeta/2-1}$ then
10             break
11         $c \leftarrow c \cup \{v\}$
12         $q \leftarrow q \cup \mathrm{FI}(v)$  ▷ Descendingly sort $\mathrm{FI}(v)$ by $\iota$
13         $v \leftarrow q.\mathrm{pop}()$
14     $P \leftarrow P \cup \{u \mid u \in \mathrm{FI}(v) \setminus c, v \in c\}$
15     $cs \leftarrow cs \cup \{c\}$
16 $\psi \leftarrow 1$
17 while $|cs| \geq |L|$ do
18     $V' \leftarrow$ Regard each $c \in cs$ as a vertex
19     $E' \leftarrow$ Create edges among $V'$ through mask $\psi$
20     $cs \leftarrow$ Perform Leiden for $G'(V', E')$ with $L$
21     $\psi \leftarrow \psi + 1$
22 return $cs$
```

B. Circuit Clustering

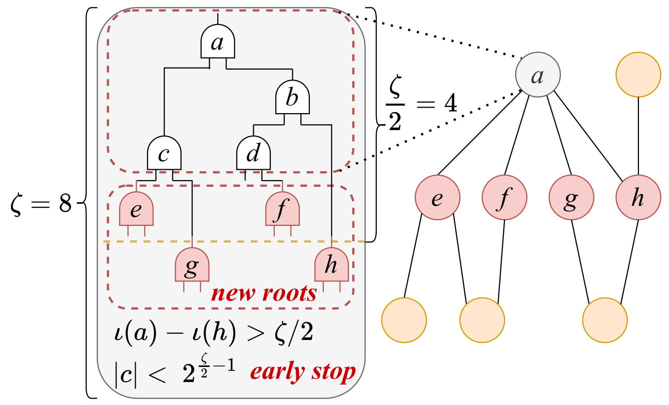
As discussed in Section II-C, conventional partition or clustering techniques often struggle to accurately determine which vertices should be grouped together to form subcircuits that facilitate efficient and effective logic optimization. These methods typically rely on purely structural metrics or heuristic community detection algorithms, which may not align well with the functional and optimization-relevant characteristics of Boolean networks. Consequently, they often generate partitions that fail to capture the true logical boundaries and interaction patterns necessary for high-quality synthesis. To address this limitation, we propose a partition algorithm specifically tailored for logic optimization in Boolean networks. Unlike traditional approaches that pursue general-purpose structural decomposition, our algorithm focuses on identifying subcircuits that are particularly conducive to logic simplification and performance improvement. The algorithm leverages both structural features and optimization potential to guide partition decisions in a manner that aligns with the requirements of synthesis and transformation tasks. The complete methodology, including algorithmic details, heuristics, and implementation considerations, is illustrated in Algorithm 2, which consists of two steps: (i) initial partition (lines 1-15) and (ii) hierarchical clustering (lines 16-21).

1) Initial Partition: Given the intricate topological and functional structure of Boolean networks, coupled with the cone-based characteristics of most logic optimization techniques, it is essential to perform a high-quality initial partition before global clustering. Such an initial decomposition not only facilitates more effective downstream optimization, but also ensures structural compatibility with cone-centric logic synthesis frameworks.

To this end, we introduce a specialized initial partitioning algorithm designed to identify structurally meaningful subgraphs. The process begins with the calculation of the logic levels ι, as defined in Section III-A, for all vertices in the Boolean network G (line 1). The algorithm then traverses the network G from its POs to effectively facilitate initial partition. Specifically, all POs are inserted into a priority queue P, which organizes the elements in descending order according to their logic levels (line 2). This sorting strategy ensures that vertices with a greater logical depth are prioritized during cone expansion, thus increasing the potential for optimization. The initial partitioning proceeds iteratively (lines 4-15). At each iteration, the vertex with the highest logic level is extracted from P and designated as the root r of a new cone c (line 5). The cone c is then expanded through a traversal toward the input direction, using a breadth-first search strategy (lines 8-13). This expansion follows the direction opposite to the circuit's signal flow, effectively tracing functional dependencies from outputs to inputs and naturally exposing the underlying logic structure.

Notably, the inclusion of a vertex v into a cone c implies that v is no longer eligible to serve as a candidate root in subsequent cone construction iterations. This constraint underscores the importance of carefully maintaining the quality of each cone, as suboptimal selections in earlier stages may compromise the overall effectiveness of the partition and optimization process. To assess the quality of a cone, we primarily consider the number of vertices it encompasses. A cone with a greater number of vertices typically reflects a denser concentration of logic elements within a constrained range of ι difference. This dense grouping implies a higher likelihood of internal logic redundancy within the cone, which in turn enhances its potential for logic simplification and optimization. Consequently, striking an appropriate balance between cone size and logic optimization potential becomes essential. To achieve this balance, an early stop mechanism is introduced (lines 9-10), which serves as a dynamic control for the cone expansion process. Specifically, as the cone expands from its root vertex r, the algorithm monitors both the ι range and the size of the current cone. If the condition $\iota(r) - \iota(v) > \zeta/2$ is met for a newly encountered vertex v, and the current cone size satisfies $|c| < 2^{\zeta/2-1}$, then the expansion is halted prematurely. This stop condition prevents overextending a cone that is too small to yield meaningful optimization benefits. Otherwise, if the condition is not satisfied, all vertices that meet the criterion $\iota(r) - \iota(v) < \zeta$ are incorporated into the cone c.

An illustrative example of this early stop strategy is shown in Fig. 4, where the threshold parameter is set to $\zeta = 8$. During the expansion of a cone rooted at vertex a, the algorithm reaches vertex h and evaluates whether to continue expansion. At this stage, the cone contains 4 vertices, which is less than the threshold $2^{\zeta/2-1} = 8$. Therefore, the expansion is terminated early, and the vertices e, f, g, and h, which lie beyond the acceptable size depth tradeoff, are added to the candidate set P for potential use as roots in future cone construction. This example highlights how the mechanism



$\zeta = 8$
$\frac{\zeta}{2} = 4$
a
b
c
d
e
f
g
h
new roots
$\iota(a) - \iota(h) > \zeta/2$
$|c| < 2^{\frac{\zeta}{2}-1}$ early stop

Fig. 4. Example of early stop mechanism.

effectively maintains both diversity in cone root selection and structural quality in logic clustering.

After expansion, the algorithm updates the priority queue P (line 14) and stores the constructed cone c in the partition set cs for subsequent processing (line 15). The algorithm terminates when the priority queue P is empty. As illustrated in the lower-right portion of Fig. 3, this procedure yields a set of five distinct clusters, each representing a localized logic region. These clusters serve as the output of the initial partition stage and lay the foundation for subsequent hierarchical clustering and sub-circuit optimization stages.

2) Hierarchical Clustering: Hierarchical clustering begins with the initial partition obtained from Section IV-B1 and progressively merges these clusters to yield high-quality clustering results (lines 16-21). ψ is a masking that will be utilized in the subsequent hierarchical clustering phase, and it gradually increases with the number of iterations (line 16). Initially, each cluster $c \in cs$ is treated as an individual vertex and added to the vertex set V' (line 18). The edges in the intermediate graph G', denoted as E', are constructed based on the connectivity information from the original graph G, with the process governed by a masking ψ (line 19). Specifically, for any pair of vertices u and v belonging to clusters c_i and c_j, respectively, an edge is established between c_i and c_j if $\rho_{u,v} > 0$, that is, their logic level difference satisfies $\iota(u) - \iota(v) \leq \psi$.

This masking strategy provides two key benefits that enhance the quality and efficiency of the subsequent partition and optimization processes. First, from the perspective of Boolean function decomposition, a significant difference in the ι values between two vertices can be interpreted as an indication of the extraction of common factors. In such scenarios, clustering these vertices within the same subgraph by retaining their connecting edge may obscure opportunities for more effective algebraic simplification. To mitigate this, the algorithm is designed to mask such edges, effectively ignoring their influence during clustering. This selective omission enables the clustering process to better explore the potential optimization. Second, from a timing or path-sensitivity perspective, vertices that reside on the critical path, or more generally on deeper logic paths, are typically more sensitive to interconnect changes. Retaining the edges between such vertices is essential for preserving the original timing characteristics and enabling delay-aware optimizations. The masking strategy, by design, preserves these structurally and temporally important edges, ensuring that clustering does not inadvertently degrade circuit performance. As a result, the clusters produced through this strategy naturally promote depth-aware logic grouping. This not only facilitates more effective optimization in subsequent logic synthesis stages but also contributes to the construction of logic networks with reduced logic depth and improved performance profiles.

In the original graph G, each vertex v is associated with a label $L(v)$, which reflects its cost information for different logic optimization algorithms. In the constructed graph G', this property must be preserved to ensure meaningful guidance during the hierarchical clustering process. Accordingly, for each vertex c in G', its label $L(c)$ is defined as follows:

$$L(c) = \frac{\Sigma_{v \in c} L(v)}{|c|}, \tag{10}$$

where $|c|$ refers to the number of vertices in G which are contained in c. For an edge (c_i, c_j) in G', its weight w_{c_i,c_j} is defined as:

$$w_{c_i,c_j} = \sum_{u \in c_i} \sum_{v \in c_j} \rho_{u,v} \times (\psi - \iota(u) + \iota(v)) + s_{c_i,c_j}, \tag{11}$$

$$s_{c_i,c_j} = \sum_{\lambda=0}^{|L|-1} \mathbb{1}(r_{c_i,\lambda}, r_{c_j,\lambda}) \times 2^{|L| - r_{c_i,\lambda}}, \tag{12}$$

where $r_{c_i,\lambda}$ represents the index value of $l_\lambda(c_i)$ in ascending order of $L(c_i)$. The indicator function $\mathbb{1}(r_{c_i,\lambda}, r_{c_j,\lambda})$ equals 1 if $r_{c_i,\lambda} = r_{c_j,\lambda}$ and equals 0 otherwise. When two vertices c_i and c_j exhibit high similarity s_{c_i,c_j}, this increases the tendency of these vertices to be grouped into the same cluster.

After constructing the graph G', the Leiden algorithm is applied to cluster the vertices within this graph (line 20). Following each clustering iteration, the masking is updated by $\psi = \psi + 1$ (line 21). Constraining ψ to a relatively small value enables the hierarchical clustering to capture variations across topological levels in a fine-grained way. This process is repeated iteratively until the number of resulting clusters falls below the specified threshold $|L|$. As illustrated in the lower part of Fig. 3, the hierarchical clustering procedure ultimately generates four distinct cluster groups, each visually distinguished by a different color.

The computational complexity analysis of Algorithm 2 reveals its efficiency advantages. During the initial partition stage, each vertex is evaluated once to determine its initial cluster assignment, resulting in a complexity of $O(|V|)$ for vertex processing and $O(|E|)$ for edge traversal (lines 4-15). In the subsequent hierarchical clustering stage, the masking filtering process depends on the number of edges in the Boolean network, yielding $O(|E|)$ complexity (line 19). Since the `Leiden` algorithm operates with linear complexity in the number of edges, this stage contributes $O(|E'|)$ where $|E'| \leq |E|$ due to masking (line 20). Assuming the algorithm performs I hierarchical iterations until convergence (lines 17-21), the overall time complexity can be expressed as $O(I \times |E|)$, where I is typically small in practice (usually $I \leq 10$ for most circuits).

Algorithm 3: Overall flow of CHOP.

```
Input: A Boolean network $G(V, E)$.
Output: An optimized circuit $G'(V', E')$.
1 $L \leftarrow$ Predict the labels of all vertices $V$
2 $cs \leftarrow$ Perform circuit clustering for $G$ with $L$
3 for $c \in cs$ do
4     $o \leftarrow$ Select the optimization method for $c$
5     $c \leftarrow$ Perform the selected optimization $o$ for $c$
6 $G'(V', E') \leftarrow$ Merge $cs$
7 return $G'(V', E')$
```

C. Sub-circuit Optimization

After circuit clustering, the Boolean network is divided into multiple clusters, each representing a localized subcircuit. To fully exploit the optimization potential of these subcircuits, it is necessary to select appropriate logic optimization strategies tailored to the structural and functional characteristics of each cluster. To this end, we propose a logic optimization selection algorithm that comprises two key components: tendency acquisition and sub-circuit optimization. The tendency acquisition module is responsible for analyzing each cluster and inferring its optimization preferences or tendencies, which may reflect a specific Boolean network structure for which the subcircuit is more suitable. Based on this guidance, the subcircuit optimization module then applies the corresponding optimization method to each cluster, ensuring that the selected strategy aligns with the intrinsic logic structure and optimization potential of the subcircuit.

1) Tendency Acquisition: Each cluster $c \in cs$ is associated with a circuit graph optimization method most suitable for it. The tendency $T(c)$ can be estimated based on the label L of the vertices within c, enabling a more targeted and effective optimization strategy, i.e.,

$$T(c) = \text{argmin } L(c). \tag{13}$$

Since $L(c)$ is generated by aggregation of $L(v), v \in c$, choosing the minimum value of $L(c)$ means that most vertices in c prefer to choose the $T(c)^{th}$ optimization algorithm. The lower left part of Fig. 3 shows an example, where the optimization method corresponding to the index marked by red digits is selected for the cluster.

2) Sub-circuit Optimization: Each cluster is subsequently optimized using its designated logic optimization technique. The optimized clusters are then merged into a complete Boolean network by aligning their PIs and POs based on their original correspondences. As a result, an optimized Boolean network is reconstructed. This process is illustrated in Fig. 3, where the final concatenation of optimized clusters into the output Boolean network is highlighted in blue.

D. Overall Flow of CHOP

In summary, the overall workflow of CHOP is illustrated in Algorithm 3. The process begins by utilizing the pre-trained model, as introduced in Section IV-A, to predict the label $L(v)$ for each vertex v within the Boolean network G (line 1). This learning-based approach effectively mitigates the inefficiency of conventional feature extraction methods. Subsequently, the network G, along with the resulting set of labels L, is fed to the circuit clustering algorithm detailed in Section IV-B to obtain the clusters cs (line 2). By incorporating both initial partition and hierarchical clustering, the algorithm is capable of generating optimization-oriented subcircuits. Finally, each cluster c undergoes logic optimization using the techniques introduced in Section IV-C (lines 3-5), and the resulting subcircuits are then merged to construct the final optimized circuit $G'(V', E')$ (line 6).

V. Experimental Results

The proposed CHOP framework was implemented in C++ and Python. For evaluation, we employed circuits from EPFL benchmarks [20]. All experiments were conducted on a machine with an Intel(R) Xeon(R) Platinum 8350C processor, an NVIDIA A100 GPU, and 1.5 TB of memory. To evaluate the effectiveness of CHOP, we employed the same four types of logic representations and their corresponding optimization methods as utilized by LSOracle and HeLO. These optimization methods involve rewrite, refactor, balance, and resubstitution, all of which are implemented in the mockturtle [37], an open-source logic network library. The baselines consist of the following two parts:

- A single logic representation along with its corresponding optimization method, including AIG, MIG, XMG, and XAG.
- Hybrid optimization frameworks, specifically LSOracle [16] and HeLO [17]. Since we were unable to obtain the prediction model of LSOracle, we performed all the logic optimization methods it provides and selected the best outcome as the final result.

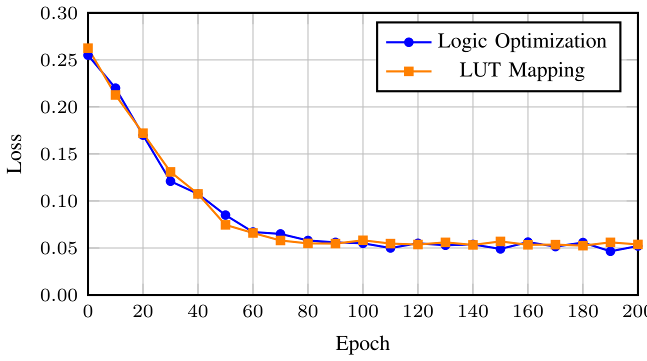


Fig. 5. Training loss curves with logic optimization and LUT mapping as the targets, respectively.

A. Accuracy of Labeling Vertices

We extracted all subgraphs using the method described in Section IV-A1. During the expansion process, each vertex was expanded with an expansion degree of $k = 3$. Afterward, we applied the four aforementioned logic optimization methods to each subgraph. We evaluated the number of vertices and the logic depth after optimization, which reflect the structure of the subgraph prior to technology mapping. We further assessed the latency and area after performing 6-LUT mapping to characterize the post-mapping implementation quality. These evaluation results were then used to compute the labels defined in Equation (6), ensuring that the assigned labels accurately reflect the behavior of each subgraph under different optimization strategies. Finally, we fed these subgraphs and their corresponding labels into the prediction model as training data.

We expanded subgraphs centered at each vertex across a set of circuits from the ISCAS'85 [38] and EPFL [20] benchmarks, and randomly sampled 10,000 subgraphs to construct the training dataset. The model was trained for 200 epochs, with the loss function defined in Section IV-A3. The training loss curve is illustrated in Fig. 5, where the loss value stabilizes after approximately 80 epochs. Using the index corresponding to the minimum predicted value of L as the evaluation criterion, the model achieves final accuracies of 99.62% and 99.38%, respectively.

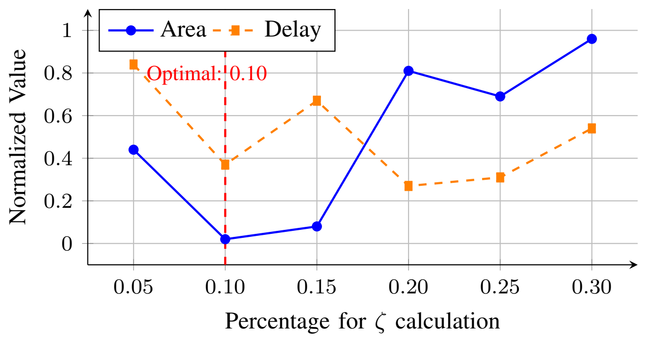


Fig. 6. Parameter sensitivity analysis for deep circuits: Impact of ζ percentage on optimization quality for the `log2` circuit (depth=444). The optimal percentage of 0.10 minimizes both area and delay deviations, validating our parameter choice $\zeta = \max(\text{depth} \times 0.1, 10)$.

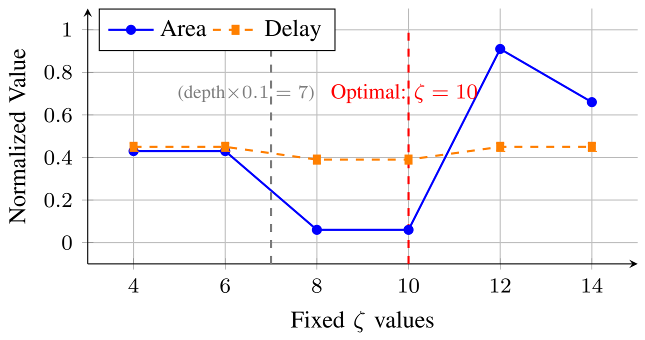


Fig. 7. Parameter sensitivity analysis for shallow circuits: Impact of fixed ζ values on optimization quality for the `voter` circuit (depth=70). The optimal value $\zeta = 10$ provides the best balance, demonstrating the effectiveness of our minimum threshold in the formula $\zeta = \max(\text{depth} \times 0.1, 10)$.

B. Setting of ζ

To rigorously evaluate the impact of the parameter ζ, we conducted controlled experiments under two complementary

TABLE II. Experimental results on the EPFL benchmark [20] after logic optimization.

Circuit	AIG			MIG			XMG			XAG			LSOracle [16]			HeLO [17]			CHOP	
	depth	node	NDP	depth	node	NDP	depth	node	NDP	depth	node	NDP	depth	node	NDP	depth	node	NDP	depth	node
adder	55	1906	4.78	14	1567	1.00	14	1567	1.00	213	1016	9.86	14	1567	1.00	12	2043	1.12	14	1567
arbiter	20	5140	1.24	10	6335	0.77	15	5695	1.03	59	8973	6.41	29	5688	2.00	10	6392	0.77	13	6354
bar	16	3599	1.63	12	3336	1.13	12	3336	1.13	12	3141	1.06	16	3599	1.63	12	3337	1.13	12	2952
cavlc	12	715	1.22	10	702	1.00	15	697	1.49	14	729	1.45	10	702	1.00	10	702	1.00	10	702
ctrl	13	96	1.61	6	172	1.33	8	180	1.86	8	137	1.41	6	124	0.96	6	172	1.33	8	97
dec	3	304	1.00	3	304	1.00	3	304	1.00	3	304	1.00	3	304	1.00	3	304	1.00	3	304
div	1713	122441	2.30	970	94088	1.00	970	94088	1.00	4372	57247	2.74	1917	117962	2.48	743	145573	1.19	970	94088
hyp	16240	320826	1.33	16116	242549	1.00	16116	242549	1.00	20717	197031	1.04	21234	230663	1.25	15964	265710	1.09	16118	242367
i2c	9	1417	0.97	11	1381	1.15	16	1358	1.65	13	1416	1.40	9	1417	0.97	11	1381	1.15	9	1465
int2float	10	258	1.25	8	258	1.00	13	258	1.63	12	254	1.48	8	258	1.00	8	258	1.00	8	258
log2	491	33044	1.65	238	40214	0.97	238	40214	0.97	366	36277	1.35	343	39109	1.36	232	73294	1.73	252	39060
max	80	6909	2.46	47	6418	1.34	47	6418	1.34	187	3739	3.11	47	6418	1.34	54	7812	1.88	60	3749
memctrl	57	79655	1.27	69	51762	1.00	95	47472	1.26	83	56188	1.31	98	54531	1.50	61	56592	0.97	69	51762
multiplier	266	25844	1.48	122	35342	0.93	122	35342	0.93	274	27062	1.59	181	31043	1.21	135	49548	1.44	149	31239
priority	210	734	1.11	126	1101	1.00	126	1101	1.00	166	1107	1.32	126	1101	1.00	126	1101	1.00	126	1101
router	16	443	1.14	13	478	1.00	16	455	1.17	31	291	1.45	13	478	1.00	13	478	1.00	13	478
sin	271	5277	1.75	112	7315	1.00	112	7315	1.00	165	6184	1.25	141	6483	1.12	126	9667	1.49	112	7315
sqrt	6230	24729	1.49	3974	26080	1.00	3974	26080	1.00	5060	17856	0.87	5072	19165	0.94	3942	31486	1.20	3974	26080
square	60	23538	1.35	38	21864	0.79	41	21289	0.83	224	18549	3.96	87	21528	1.78	53	22335	1.13	50	20997
voter	76	9608	1.10	59	14002	1.24	60	13990	1.26	62	11760	1.09	76	10698	1.22	68	15292	1.56	72	9252
Ave. ratio	1.53	1.07	1.61	0.94	1.11	1.03	1.09	1.10	1.18	2.61	0.96	2.26	1.24	1.06	1.29	0.95	1.28	1.21	1.00	1.00

NDP: normalized node-depth product relative to CHOP; depth: logic depth; node: number of nodes.

scenarios, designed to analyze its effect across circuits with different logic depths.

For deep logic circuits, we select the `log2` circuit (logic depth: 444, 32,060 nodes before optimization), where ζ is determined by scaling the logic depth with varying percentages. This allows us to examine how different scaling factors influence optimization outcomes. For shallow logic circuits, we use the `voter` circuit (logic depth: 70, 13,758 nodes before optimization), where ζ is evaluated both as a proportional parameter and as fixed values. This dual approach enables a comprehensive analysis of ζ's sensitivity under different depth constraints. The experimental results are illustrated in Fig. 6 and Fig. 7.

The analysis reveals distinct optimization patterns across both scenarios. As shown in Fig. 6, setting the scaling percentage to 0.10 for deep circuits uniquely achieves both area and delay reductions below the mean, yielding the lowest area-delay product among all tested configurations. This indicates that excessively small ζ values underutilize the clustering potential by creating overly fine initial partitions, while excessively large values generate coarse partitions that limit subsequent refinement opportunities.

Similar trends emerge for shallow circuits (Fig. 7), where the optimal fixed value $\zeta = 10$ provides superior balance between clustering granularity and optimization effectiveness. To ensure robust performance across diverse circuit depths, we establish a minimum threshold of 10, leading to the final parameter configuration: $\zeta = \max(\text{depth} \times 0.1, 10)$.

C. *Results of Logic Optimization*

TABLE II presents the experimental results after logic optimization on the EPFL benchmark, to minimize the node-depth product (NDP), *i.e.*, the product of the number of nodes ("node") and logic depth ("depth"). Notably, the NDP in TABLE II is the ratio of the NDP of each baseline to the NDP of CHOP. The results indicate that the optimization performance varies significantly across the four single logic representation methods. In most cases, the MIG-based method yields superior performance among these four methods. These findings underscore the limitations of existing logic optimization techniques that focus solely on a single logic representation. In addition, although XMG has the highest expressive power among these four logic representations, the results reveal that XMG-based optimization techniques do not consistently outperform their counterparts. This inconsistency may stem from the heuristic nature of the current optimization algorithms, which may not fully leverage the expressive capabilities of XMG.

Furthermore, hybrid optimization methods also exhibit varying performance outcomes. Notably, in certain cases, the overall performance of the hybrid strategy does not surpass that of applying MIG alone. This phenomenon may result from the relatively strong performance of MIG during the logic optimization phase, while many subgraphs within the hybrid method may have missed the opportunity to adopt MIG as the final optimization method. Overall, CHOP shows the best performance, specifically with average NDP reductions by 28.79%, 1.34%, 11.10%, 33.38%, 16.44%, and 13.33% when compared with AIG, MIG, XMG, XAG, LSOracle, and HeLO, respectively. These significant improvements highlight the importance of the combination of circuit clustering and optimization method selection.

D. *Results of LUT Mapping*

However, it is worth noting that logic-level optimization metrics alone do not fully capture the practical implications for physical design. This is because different vertex types in a Boolean network not only differ in expressive power but also require varying amounts of physical resources to implement. As a result, logic-level metrics such as gate count and logic depth provide only a partial view of overall design quality. To more accurately assess the practical effectiveness of the

TABLE III. Experimental results on the EPFL benchmark [20] after 6-LUT mapping.

Circuit	AIG			MIG			XMG			XAG			LSOracle [16]			HeLO [17]			CHOP	
	delay	area	ADP	delay	area	ADP	delay	area	ADP	delay	area	ADP	delay	area	ADP	delay	area	ADP	delay	area
adder	64	192	2.81	11	397	1.00	11	397	1.00	64	192	2.81	11	397	1.00	9	583	1.20	11	397
arbiter	9	1876	1.21	9	2577	1.67	9	2577	1.67	23	2313	3.82	12	1877	1.62	9	3439	2.22	8	1741
bar	4	512	1.00	4	512	1.00	4	512	1.00	4	512	1.00	4	512	1.00	4	512	1.00	4	512
cavlc	8	114	1.31	6	119	1.03	6	119	1.03	6	119	1.03	8	114	1.31	6	150	1.29	6	116
ctrl	2	28	1.00	2	28	1.00	2	28	1.00	2	28	1.00	2	28	1.00	2	30	1.07	2	28
dec	2	273	1.00	2	273	1.00	2	273	1.00	2	273	1.00	2	273	1.00	2	273	1.00	2	273
div	2005	4213	1.01	776	27491	2.54	776	27491	2.54	1606	25529	4.88	1915	11869	2.71	732	28889	2.52	2010	4180
hyp	8520	49736	1.00	8520	49736	1.00	8520	49736	1.00	8520	49736	1.00	8509	51437	1.03	8651	69321	1.42	8520	49736
i2c	5	342	1.13	5	342	1.13	5	342	1.13	5	342	1.13	5	342	1.13	5	362	1.20	5	302
int2float	4	47	1.00	4	47	1.00	4	47	1.00	4	47	1.00	4	47	1.00	4	56	1.19	4	47
log2	182	7893	1.24	129	9102	1.01	129	9103	1.01	185	8125	1.30	151	8720	1.14	137	22455	2.66	143	8094
max	45	1689	2.49	22	2288	1.65	22	2288	1.65	89	787	2.29	45	1689	2.49	29	2574	2.44	34	899
memctrl	58	12064	1.30	45	11969	1.00	45	11969	1.00	49	11709	1.07	49	12657	1.15	45	11969	1.00	45	11969
multiplier	127	6348	1.77	68	7453	1.12	76	6821	1.14	119	6907	1.81	91	6919	1.39	76	9533	1.59	75	6058
priority	35	225	1.00	33	264	1.11	33	265	1.11	33	265	1.11	35	225	1.00	116	391	5.76	35	225
router	13	53	1.00	13	53	1.00	13	53	1.00	13	53	1.00	13	53	1.00	8	110	1.28	13	53
sin	95	1590	1.27	66	1925	1.07	66	1925	1.07	83	1583	1.10	77	1714	1.11	81	2708	1.84	67	1779
sqrt	2034	4086	1.12	3897	8141	4.29	3897	8141	4.29	3689	7839	3.91	1957	3992	1.06	3701	10664	5.34	1930	3828
square	48	4992	2.12	30	4822	1.28	32	4559	1.29	122	4130	4.46	56	4447	2.21	41	4830	1.75	25	4515
voter	29	2008	1.69	35	2827	2.88	35	2827	2.88	34	3328	3.29	28	1958	1.60	39	3150	3.57	22	1562
Ave. ratio	1.43	1.05	1.37	1.04	1.52	1.44	1.06	1.51	1.46	1.73	1.36	2.00	1.16	1.17	1.35	1.18	1.90	2.07	1.00	1.00

ADP: normalized area-delay product relative to CHOP; delay: circuit delay; area: circuit area.

optimization methods, TABLE III presents the results after 6-LUT mapping [39], which offers a closer approximation of the real-world performance in the implementation of integrated circuits. The objective of 6-LUT mapping is to minimize the ADP, *i.e.*, the product of circuit area and delay. Similarly, the ADP values in TABLE III are normalized for comparison. Notably, consistent with the results in TABLE II, the MIG-based method achieves the best optimization outcomes among the four single logic representation methods. The difference is that hybrid optimization methods show better results in 6-LUT mapping, especially LSOracle and CHOP.

Overall, CHOP continues to maintain a leading position relative to the baselines. Specifically, CHOP achieves ADP reductions of 19.49%, 17.28%, 17.43%, 30.16%, 17.99%, and 36.42% on average compared with AIG, MIG, XMG, XAG, LSOracle, and HeLO, respectively. In addition, there are two primary reasons why HeLO performs poorly in 6-LUT mapping. First, the framework targets logic optimization, and the input of technology mapping in [17] directly sources from the results of logic optimization. Second, HeLO's circuit partitioning method has significant shortcomings. Specifically, unlike CHOP, HeLO's initial partitioning method extends from POs to PIs without considering the situation of other cones. This makes HeLO's optimization results depend to a certain extent on the order of processing POs during the initial partitioning.

Notably, in circuits where LSOracle demonstrates suboptimal performance, CHOP consistently excels. This discrepancy arises because LSOracle aggregates edges with higher weights for partition, which can sometimes lead to the grouping of vertices with significantly different logic levels, potentially resulting in suboptimal performance in certain circuits. This observation further validates our partition method presented in Section IV-B2. This disparity further motivates our approach: rather than relying on a single optimization method across all designs, we aim to tailor the optimization strategy to the specific characteristics of each circuit or subgraph cluster. By leveraging structural and functional diversity within the design, our method seeks to achieve a better trade-off between optimization quality and generalization.

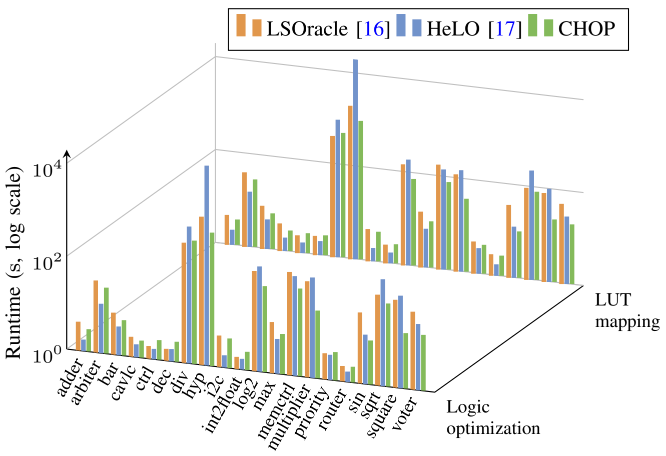


Fig. 8. Comparison of normalized runtime among different optimization methods for logic optimization and 6-LUT mapping, respectively.

E. Runtime

The runtime comparison between CHOP and two state-of-the-art hybrid optimization methods is presented in Fig. 8. On smaller circuits, CHOP does not exhibit a noticeable speed advantage, primarily because the selected model introduces a fixed runtime overhead that dominates when the circuit size is small. This overhead mainly stems from the cost of subgraph extraction, feature encoding, and inference time of the learning-based components. However, as the circuit size increases, these fixed costs become less significant relative to the overall runtime, and the scalability of CHOP becomes more evident.

Overall, CHOP shows high efficiency, with average speeds of 1.92x that of LSOracle and 2.98x that of HeLO for logic optimization, as well as 1.78x and 2.48x for LUT mapping. This substantial reduction in runtime highlights CHOP's ability to efficiently handle large-scale designs. Furthermore, since CHOP relies on pre-trained models for inference rather than costly iterative optimization, its performance is more stable and predictable across different circuit instances. These results collectively demonstrate the runtime efficiency and scalability of CHOP, making it particularly suitable for large and complex circuits where traditional methods may struggle with high computational overhead.

VI. Conclusion

This paper proposed CHOP, a novel framework that fundamentally advances hybrid logic synthesis through vertex-level feature extraction and optimization-aware clustering. The key innovation lies in shifting from the traditional "partition-then-predict" paradigm to a "predict-then-partition" approach, enabling more effective alignment between circuit structures and optimization strategies. CHOP's hierarchical clustering algorithm, guided by both structural and functional features, demonstrates superior performance across multiple metrics.

The experimental results validate CHOP's effectiveness. For logic optimization on the EPFL benchmark, CHOP achieved average NDP reductions of 16.44% and 13.33% compared with LSOracle and HeLO, with 1.92x and 2.98x speedup, respectively. For 6-LUT mapping, CHOP achieved average ADP reductions of 17.99% and 36.42%, with 1.78x and 2.48x speedup, respectively.

Future work will focus on extending CHOP to handle larger industrial circuits, integrating more sophisticated neural architectures for vertex feature extraction, and exploring applications to other EDA tasks such as placement and routing optimization.

References

[1] E. Testa, M. Soeken, L. G. Amar, and G. De Micheli, "Logic synthesis for established and emerging computing," *Proceedings of the IEEE*, vol. 107, no. 1, pp. 165–184, 2019.

[2] R. Brayton, R. Rudell, A. Sangiovanni-Vincentelli, and A. Wang, "MIS: A multiple-level logic optimization system," *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems (TCAD)*, vol. 6, no. 6, pp. 1062–1081, 1987.

[3] L. Hellerman, "A catalog of three-variable or-invert and and-invert logical circuits," *IEEE Transactions on Electronic Computers*, vol. EC-12, no. 3, pp. 198–223, 1963.

[4] L. Amarú, P.-E. Gaillardon, and G. De Micheli, "Majority-inverter graph: A novel data-structure and algorithms for efficient logic optimization," in *ACM/IEEE Design Automation Conference (DAC)*, 2014, p. 1–6.

[5] G. Meuli, M. Soeken, and G. Micheli, "Xor-And-Inverter graphs for quantum compilation," *npj Quantum Information*, vol. 8, 12 2022.

[6] W. Haaswijk, M. Soeken, L. Amarù, P.-E. Gaillardon, and G. De Micheli, "A novel basis for logic rewriting," in *IEEE/ACM Asia and South Pacific Design Automation Conference (ASPDAC)*, 2017, pp. 151–156.

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

[8] H. Riener, W. Haaswijk, A. Mishchenko, G. Micheli, and M. Soeken, "On-the-fly and DAG-aware: Rewriting boolean networks with exact synthesis," in *IEEE/ACM Proceedings Design, Automation and Test in Eurpoe (DATE)*, 03 2019, pp. 1649–1654.

[9] A. T. Calvino and G. De Micheli, "Scalable logic rewriting using don't cares," in *IEEE/ACM Proceedings Design, Automation and Test in Eurpoe (DATE)*, 2024, pp. 1–6.

[10] S.-Y. Lee and G. D. Micheli, "Heuristic logic resynthesis algorithms at the core of peephole optimization," *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems (TCAD)*, vol. 42, no. 11, pp. 3958–3971, 2023.

[11] L. Amarú, M. Soeken, P. Vuillod, J. Luo, A. Mishchenko, J. Olson, R. Brayton, and G. De Micheli, "Improvements to boolean resynthesis," in *IEEE/ACM Proceedings Design, Automation and Test in Eurpoe (DATE)*, 2018, pp. 755–760.

[12] A. M. R. Brayton and A. Mishchenko, "Scalable logic synthesis using a simple circuit structure," in *IEEE/ACM International Workshop on Logic Synthesis*, vol. 6, 2006, pp. 15–22.

[13] H. Riener, E. Testa, L. Amaru, M. Soeken, and G. De Micheli, "Size optimization of MIGs with an application to QCA and STMG technologies," in *IEEE/ACM International Symposium on Nanoscale Architectures (NANOARCH)*, 2018, p. 157–162.

[14] S.-Y. Lee, H. Riener, A. Mishchenko, R. K. Brayton, and G. De Micheli, "A simulation-guided paradigm for logic synthesis and verification," *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems (TCAD)*, vol. 41, no. 8, pp. 2573–2586, 2022.

[15] L. Amarú, P.-E. Gaillardon, and G. De Micheli, "MIXSyn: An efficient logic synthesis methodology for mixed XOR-AND/OR dominated circuits," in *IEEE/ACM Asia and South Pacific Design Automation Conference (ASPDAC)*, 2013, pp. 133–138.

[16] W. L. Neto, M. Austin, S. Temple, L. Amaru, X. Tang, and P.-E. Gaillardon, "LSOracle: a logic synthesis framework driven by artificial intelligence: Invited paper," in *IEEE/ACM International Conference on Computer-Aided Design (ICCAD)*, 2019.

[17] Y. Pu, F. Liu, Z. He, K. Zhu, R. Fu, Z. Wang, T.-Y. Ho, and B. Yu, "HeLO: A heterogeneous logic optimization framework by hierarchical clustering and graph learning," in *ACM International Symposium on Physical Design (ISPD)*, 2025, p. 116–124.

[18] R. Fu, R. Zhang, Z. Zheng, Z. Shi, Y. Pu, J. Huang, Q. Xu, and T.-Y. Ho, "Late breaking results: Hybrid logic optimization with predictive self-supervision," in *ACM/IEEE Design Automation Conference (DAC)*, 2025.

[19] S. Schlag, T. Heuer, L. Gottesbüren, Y. Akhremtsev, C. Schulz, and P. Sanders, "High-quality hypergraph partitioning," *ACM Journal of Experimental Algorithmics*, vol. 27, 2023.

[20] L. Amarù, P.-E. Gaillardon, and G. De Micheli, "The EPFL combinational benchmark suite," in *IEEE/ACM International Workshop on Logic Synthesis*, 2015.

[21] S. Kauffman, "Metabolic stability and epigenesis in randomly constructed genetic nets," *Journal of Theoretical Biology*, vol. 22, no. 3, pp. 437–467, 1969.

[22] S. Fortunato, "Community detection in graphs," *Physics Reports*, vol. 486, no. 3, pp. 75–174, 2010.

[23] M. E. J. Newman and M. Girvan, "Finding and evaluating community structure in networks," *Physical Review E*, vol. 69, p. 026113, 2004.

[24] V. D. Blondel, J.-L. Guillaume, R. Lambiotte, and E. Lefebvre, "Fast unfolding of communities in large networks," *Journal of Statistical Mechanics: Theory and Experiment*, vol. 2008, no. 10, p. P10008, 2008.

[25] V. A. Traag, L. Waltman, and N. J. van Eck, "From Louvain to Leiden: guaranteeing well-connected communities," *Scientific Reports*, vol. 9, no. 5233, pp. 2045–2322, 2019.

[26] 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.

[27] 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.

[28] Z. Shi, M. Li, S. Khan, L. Wang, N. Wang, Y. Huang, and Q. Xu, "Deeptpi: Test point insertion with deep reinforcement learning," in *IEEE International Test Conference (ITC)*. IEEE, 2022, pp. 194–203.

[29] S. Khan, Z. Shi, M. Li, and Q. Xu, "DeepSeq: Deep sequential circuit learning," *arXiv preprint arXiv:2302.13608*, 2023.

[30] M. Li, Z. Shi, Q. Lai, S. Khan, S. Cai, and Q. Xu, "On EDA-driven learning for SAT solving," in *ACM/IEEE Design Automation Conference (DAC)*. IEEE, 2023.

[31] Z. Shi, T. Tang, S. Khan, H.-L. Zhen, M. Yuan, Z. Chu, and Q. Xu, "EDA-driven preprocessing for SAT solving," *arXiv preprint arXiv:2403.19446*, 2024.

[32] 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.

[33] 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," *arXiv preprint arXiv:2403.01317*, 2024.

[34] 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 *IEEE/ACM International Conference on Computer-Aided Design (ICCAD)*, 2024.

[35] Z. Shi, Z. Zheng, S. Khan, J. Zhong, M. Li, and Q. Xu, "DeepGate3: Towards scalable circuit representation learning," in *IEEE/ACM International Conference on Computer-Aided Design (ICCAD)*, 2025.

[36] 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.

[37] M. Soeken, H. Riener, W. Haaswijk, E. Testa, B. Schmitt, G. Meuli, F. Mozafari, S.-Y. Lee, A. Tempia Calvino, and G. Marakkalage, Dewmini Sudara De Micheli, "The EPFL logic synthesis libraries," 2022, arXiv:1805.05121v3.

[38] M. Hansen, H. Yalcin, and J. Hayes, "Unveiling the ISCAS-85 benchmarks: a case study in reverse engineering," *IEEE Design & Test of Computers*, vol. 16, no. 3, pp. 72–80, 1999.

[39] G. Liu and Z. Zhang, "A parallelized iterative improvement approach to area optimization for LUT-based technology mapping," in *ACM International Symposium on Field-Programmable Gate Arrays (FPGA)*, 2017.

Rongliang Fu received his BS degree in software engineering from the Northwestern Polytechnical University, Xi'an, China, in 2018 and his MS degree in computer science and technology from the University of Chinese Academy of Sciences, Beijing, China, in 2021. He is currently studying for his Ph.D degree in the Department of Computer Science and Engineering, The Chinese University of Hong Kong. His research interests include electronic design automation and computer architecture.

Ran Zhang received his B.S. degree from Northeastern University, Shenyang, in 2023. He is currently a master's student at the University of Chinese Academy of Sciences (UCAS), under the Institute of Computing Technology, Chinese Academy of Sciences. His research interests include electronic design automation and computer architecture.

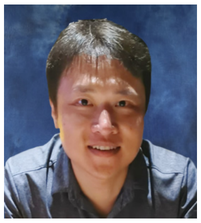
Ziyang Zheng (Student Member, IEEE) is currently a third-year Ph.D. student in the CURE Lab at The Chinese University of Hong Kong, where he is advised by Prof. Qiang Xu. He received his Bachelor's degree in Data Science and Big Data from Harbin Institute of Technology (Shenzhen), China. His research interests include Multimodal Learning, Diffusion Models, and AI applications in Electronic Design Automation (EDA).

Zhengyuan Shi is a final-year PhD candidate at The Chinese University of Hong Kong. His research interests include AI for EDA and large circuit models. He has published over 20 papers in leading EDA conferences, including DAC and ICCAD, and received Best Paper nomination awards at DAC 2022 and ASP-DAC 2025. He got his B.Eng. degree with the Presidential Scholarship from Shandong University in 2021.

Yuan Pu received the B.S. degree in computer science from The Chinese University of Hong Kong, Hong Kong, in 2022. He is now a second-year Ph.D. Student at the Department of Computer Science and Engineering, the Chinese University of Hong Kong (CUHK), supervised by Prof. Bei YU since 2023 Fall. His research interest includes Combinatorial Algorithms/AI in EDA.

Junying Huang received her Ph.D. degree in microelectronics and solid-state electronics from the University of Chinese Academy of Sciences in 2016. Currently she is an associate professor with the Department of High-throughput Computer Research Center, Institute of Computing Technology, Chinese Academy of Sciences. Her research interests include superconductive RSFQ logic, computer architecture, electronic design automation, and hardware security.

Bei Yu (M'15-SM'22) received the Ph.D. degree from The University of Texas at Austin in 2014. He is currently a Professor in the Department of Computer Science and Engineering, The Chinese University of Hong Kong. He has served as TPC Chair of ACM/IEEE Workshop on Machine Learning for CAD, and in many journal editorial boards and conference committees. He received ten Best Paper Awards from IEEE TSM 2022, DATE 2022, ICCAD 2021 & 2013, ASPDAC 2021 & 2012, ICTAI 2019, Integration, the VLSI Journal in 2018, ISPD 2017, SPIE Advanced Lithography Conference 2016, and many other awards, including DAC Under-40 Innovator Award (2024), IEEE CEDA Ernest S. Kuh Early Career Award (2022), and Hong Kong RGC Research Fellowship Scheme (RFS) Award (2024).



Qiang Xu (Senior Member, IEEE) received the B.E. and M.E. degrees from Beijing University, Beijing, China, in 1997 and 2000, respectively, and the Ph.D. degree from McMaster University, Hamilton, ON, Canada, in 2005. He is a Professor of Computer Science and Engineering with The Chinese University of Hong Kong, Hong Kong. His research interests include electronic design automation, trusted computing, and representation learning. He is currently serving as an Associate Editor for IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems and Integration, the VLSI Journal.

Tsung-Yi Ho (F'24) is a Professor in the Department of Computer Science and Engineering, The Chinese University of Hong Kong (CUHK). He received his Ph.D. in Electrical Engineering from National Taiwan University in 2005. His research interests include several areas of computing and emerging technologies, especially in the design automation of microfluidic biochips. He was a recipient of the Best Paper Award at the IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems in 2015. Currently, he serves as the VP Conferences of IEEE CEDA, and the Executive Committee of ASP-DAC and ICCAD. He is a Distinguished Member of ACM and a Fellow of IEEE.