AiLO: A Predictive Framework for Logic Optimization Using Multi-Scale Cross-Attention Transformer

YE CAI, Shenzhen University, China
RUI WANG, Shenzhen University, China
LIWEI NI, Institute of Computing Technology, Chinese Academy of Sciences, China
MIAO LIU, University of Chinese Academy of Sciences, China
XINGYU MENG, Pengcheng Laboratory, China
XIAOZE LIN, Pengcheng Laboratory, China
JUNFENG LIU, Pengcheng Laboratory, China
BIWEI XIE, Institute of Computing Technology, Chinese Academy of Sciences, China
XINGQUAN LI*, Pengcheng Laboratory, China

Logic Optimization (LO) is a critical stage in the chip design process, focused on improving the Quality of Results (QoR) by optimizing circuit designs to minimize area and delay. During logic optimization, evaluating the QoR after each iteration requires completing logic optimization and technology mapping. The evaluation process is highly time-consuming, restricting the number of optimization iterations possible within a given time. To address this, the AI-aided logic optimization framework (AiLO) is developed to explore more optimization operator sequences (recipes). AiLO framework consists of two core components: AI-based metric evaluation and optimization exploration. To achieve accurate evaluation, different prediction models can be integrated. A multi-scale cross-attention Transformer (CrossLO) is introduced to simulate the optimization structure of recipes across circuit at various scales to enhance the prediction accuracy. Moreover, the AI evaluation module can effectively maintain the recipe ranking, even when prediction accuracy is biased. The logic optimization exploration algorithm integrated with CrossLO (AI evaluation) shows an average improvement of 14.75% over the initial version. NSGA-II (optimization module) integrated with CrossLO achieves a significant lead over other algorithms in the same time. In addition, the AiLO framework continues to grow with the performance of the two components, demonstrating strong adaptability and flexibility.

CCS Concepts: • **Hardware → Logic synthesis**.

Additional Key Words and Phrases: Logic Synthesis, Logic Optimization, Evaluation and Optimization, Graph Neural Network, Transformer

*Corresponding author

This work is supported in part by the Major Key Project of PCL (No. PCL2023A03) and the NSF of Fujian Province under Grants (No. 2024J09045).

Authors' addresses: Ye Cai, caiye@szu.edu.cn, College of Computer Science and Software Engineering, Shenzhen University, Shenzhen, Guangdong, China; Rui Wang, 2300271050@email.szu.edu.cn, College of Computer Science and Software Engineering, Shenzhen University, Shenzhen, Guangdong, China; Liwei Ni, nlwmode@gmail.com, Institute of Computing Technology, Chinese Academy of Sciences, Beijing, China, nlwmode@gmail.com; Miao Liu, University of Chinese Academy of Sciences, Beijing, China; Xingyu Meng, Pengcheng Laboratory, Shenzhen, Guangdong, China; Xiaoze Lin, Pengcheng Laboratory, Beijing, China; Junfeng Liu, Pengcheng Laboratory, Shenzhen, Guangdong, China; Biwei Xie, Institute of Computing Technology, Chinese Academy of Sciences, Beijing, China; Xingquan Li, Pengcheng Laboratory, Shenzhen, Guangdong, China, fzulxq@gmail.com.

ACM 1084-4309/2025/XX-ARTXXX
https://doi.org/XXXXXXX.XXXXXXX

ACM Reference Format:
Ye Cai, Rui Wang, Liwei Ni, Miao Liu, Xingyu Meng, Xiaoze Lin, Junfeng Liu, Biwei Xie, and Xingquan Li. 2025. AiLO: A Predictive Framework for Logic Optimization Using Multi-Scale Cross-Attention Transformer. *ACM Trans. Des. Autom. Electron. Syst.* XX, X, Article XXX (XX 2025), 28 pages. https://doi.org/XXXXXXX.XXXXXXX

1 INTRODUCTION

In order to achieve more integration and functionalities, modern hardware designs are undergoing steep increases in their complexity and processing cost. As a result, the demand for automation and intelligence in EDA tools escalates, particularly in the domain of Logic Optimization (LO). LO is a critical step that translates High-Level Hardware Description Languages (HDL) into optimized gate-level netlists, targeting the optimization of key performance metrics such as area, power, and delay to enhance the Quality of Results (QoR) [33]. Engineers apply structural optimization transformations [23, 40] to develop optimization sequences, termed as "recipes" in LO. However, predefined optimization recipes often lack generalizability, and the search space for such recipes grows exponentially [24, 25, 37]. Traditional EDA tools, while capable of providing precise QoR assessments, are usually time-consuming and require significant manual intervention for Design Space Exploration (DSE) [48]. Moreover, the complexity and technological scale of hardware systems are rapidly expanding, which further extends the time-to-market. To address the need for hardware development and productivity enhancement, the industry increasingly seeks EDA tools with advanced intelligence to enable not only rapid and accurate QoR assessments but also efficient optimization. Therefore, a solution is urgently needed to efficiently identify the best optimization recipes.

The application of deep learning techniques, particularly Graph Neural Networks (GNNs) [34] for graph-structured data, opens new research avenues for predicting the QoR of LO recipes. GNNs are effective in extracting and learning structural features from circuit graphs, while Transformers excel at capturing and representing the intrinsic characteristics of optimization recipes [36, 50]. By integrating both GNNs and Transformers through a joint learning strategy, our approach aims to predict the QoR of previously unseen circuit optimization recipe pairs. It not only significantly enhances prediction accuracy, but also minimizes the trial-and-error process for engineers, accelerating the design workflow and providing a robust theoretical foundation and technical support for optimization recipe exploration.

Recent researches have facilitated the integration and application of Machine Learning (ML) technology in the field of EDA [35], and provided innovative approaches to the automation of LO processes. Despite significant advancements, constructing more efficient design space exploration strategies, enhancing model generalization capabilities, and reducing reliance on extensive sample data remain the primary challenges in current research in this domain. We introduce a novel deep learning network framework that simulates the dynamic changes in graph embeddings and optimization recipe structures across different scales. When faced with unknown optimization recipes, it significantly improves the prediction accuracy of QoR and demonstrates predictive generalization. When tested on the Open Core and EPFL datasets in the "seen IC, Unseen recipes" scenario, CrossLO outperforms the OpenABC baseline by an average of 35.62% and 49.42% in area and delay prediction accuracy, respectively, and improves the rank correlation of optimization recipes by 121.31% and 232.00%. Ablation experiments also demonstrate that CrossLO model has strong generalization capabilities. Furthermore, the AI evaluation module is achieved by deploying well-trained deep learning neural network models. Our experiments confirm that deep learning models are capable of maintaining the ranking of recipes effectively, despite potential deviations in predictive accuracy. By CrossLO evaluation, the top 10% were selected from a recipe that was

promoted from a random 10% to 40.46%. With this finding, we develop an AI-aided Logic Optimization framework (AiLO), which guides the search path of existing methods, significantly reduces the computational and temporal costs required to obtain feedback from EDA tools. The main contributions are summarized as follows:

- **AiLO Framework**: We integrate the AI evaluation recipe framework into recipe exploration method to generate an AI-assisted logic optimization framework (AiLO). The AiLO framework can be upgraded with component replacement to improve detection performance and quality. The experiment proves that AiLO framework has high flexibility and adaptability.
- **CrossLO model**: We proposes CrossLO, a multi-scale cross-attention Transformer, integrating the complementary strengths of GNNs and Transformer models. It can effectively capture local features and global information within circuit graph embeddings and synthetic optimization recipes. In the "seen IC, unseen recipes" scenario, compared with the model without multi-scale cross-attention Transformer, the prediction accuracy and sequence rank correlation are improved by 28.02% and 107.69%, respectively.
- **AI Evaluation Module**: The rapid evaluation of optimization recipes is achieved by deploying well-trained deep-learning neural network models. Experiments confirm that AI evaluation is capable of maintaining the ranking of recipes effectively, despite potential deviations in predictive accuracy. By integrating the AI evaluation module with CrossLO, the selection of the top 10% of optimization recipes increases from random 10% to 40.46%. This result offers a novel perspective for the exploration of LO.
- **Logic Optimization Promotion**: We integrate CrossLO with reinforcement learning (RL) DRiLLS, heuristic algorithm (HA) NSGA-II and Bayesian algorithm (BO) BOiLS to assist search algorithm in logic optimization. On the basis of limiting the real evaluation of 100 recipes, QoR improved by an average of 14.75% compared to the original algorithm without integrated AI evaluation. The NSGA-II with integrated AI evaluation achieves an average 7.80% improvement over BOiLS and reduces the average run time from 8.89 hours to 11.19 minutes. With a limited optimization time of 5 hours, the NSGA-II integrated AI evaluation achieves a fault-like lead with a 25.16% QoR improvement.

The remainder of this paper is organized as follows: A systematic review of relevant research findings from existing literature is presented in the Section 2. Subsequently, Section 3 provides a detailed exposition of the research approach and methodology. In the Section 4, a series of experiments are designed to validate the feasibility and efficacy of the proposed methods, followed by a compilation and analysis of the experimental data. Section 5 summarizes the core contributions of the paper and offers perspectives on future research directions.

2 PRELIMINARIES

2.1 Logic Synthesis

Logic Synthesis is a critical phase in the design of integrated circuits (IC), which can be divided into three fundamental stages: Translation, Logic Optimization, and Technology Mapping [20, 29].

Translation involves converting Register Transfer Level (RTL) code into an unoptimized gate-level Boolean description with basic logic elements such as AND gates, OR gates, flip-flops, and latches. They are typically represented in the Directed Acyclic Graph (DAG) format, specifically the And-Inverter Graph (AIG) format [2] $G = (V, E) \in \mathbb{G}$. Nodes V in an AIG are categorized as PI, PO, and AND gates, with edges E being either buffers or inverters. This format aids in identifying and optimizing the critical paths of the circuit and is essential for subsequent physical design stages, providing a simplified approach for the expression and manipulation of Boolean functions. This

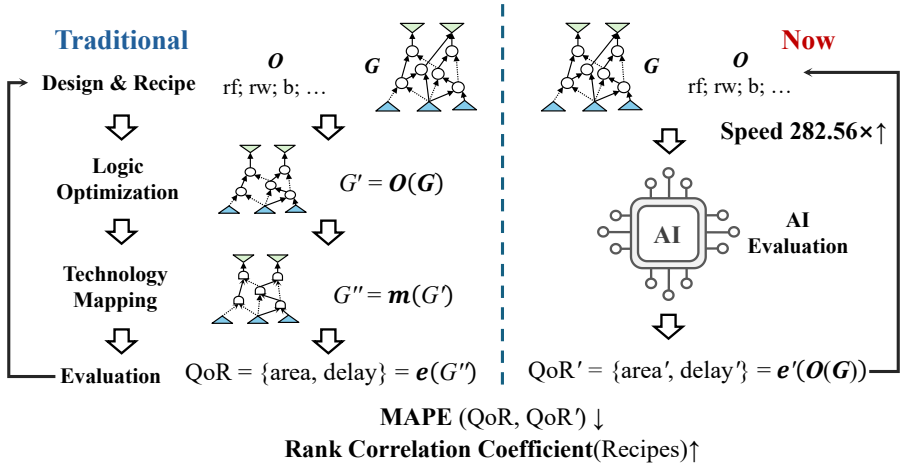


Fig. 1. Traditional Logic Synthesis Process and AI Evaluation

structure not only facilitates the implementation of LO tasks but also lays the groundwork for the subsequent mapping step [30].

The objective of the Logic Optimization stage is to transform the unoptimized Boolean description into an optimized one that meets specific design constraints related to performance, area, and power consumption. This step is crucial within the logic synthesis process, aiming to ensure the correctness of logical functionalities while optimizing circuit performance metrics. A series of structural transformation optimization operators are defined during this phase, enabling designers to flexibly select and combine different optimization strategies according to specific design objectives and constraints through a combination of serialization methods. To achieve logical optimization, a series of optimization operators is defined as $LO = [O_1, O_2, \ldots, O_k] \in \mathcal{LO}$, where k is the number of optimization operators. Optimization operators generate optimization recipes by permutation combinations, $R = \{r_1, r_2, \ldots, r_n\} \in \mathbb{R}$ be the recipes of optimization transformations applied to G, $\mathbb{R}$ represents the complete collection of recipes. Each transformation r_i in R modifies the graph G, yielding a new graph G'. This process is mathematically expressed as: $G' = O(G)$.

Technology Mapping converts the optimized Boolean description into a gate-level netlist $G'' = m(G')$. This process leverages standard cells and logic and timing information from the technology library, ensuring the final gate-level netlist meets the QoR requirements. As a critical step in the design flow, This stage significantly influences both manufacturing cost and chip performance.

2.2 Motivation

In this paper, we focus on logic optimization, which is a very crucial step for improving the QoR of circuit design. To obtain a good logic optimization result, we will run a series of logic optimization operator sequences (recipes) and select a desirable recipe with optimal QoR metrics. Traditionally, evaluating the QoR of a netlist produced by a recipe is very time-consuming. The reason is that, for a given circuit graph and a recipe, we need to run logic optimization and technology mapping, and then calculate the QoR of the mapped netlist to evaluate the effect of the recipe. As shown in Fig. 1, given a circuit G and a recipe O, then we will obtain $G' = O(G)$ after performing logic optimization, and then we will obtain $G'' = m(G')$ after performing technology mapping, and then we can obtain the area and delay of the netlist G'' by calling area and delay calculation engine. The evaluation time of traditional process on 500 optimization recipes is about 51.99 minutes.

Table 1. Logical Optimization Explores Work Comparison

	DRiLLS [18]	**BOiLS** [13]	**BSBO** [11]	**GraphML** [14]	**Yang et al.** [47]	**ABC-RL** [7]	**Zhu et al.** [51]	**EasySO** [49]	**ReLS** [22]	**AiLO(Our)**
Main Method	RL	BO	BO	GNN DL RL	GNN DL RL	GNN RL	GNN MDP RL	GNN MDP RL	GNN FC layer Explorer	GNN Transformer Explorer
Evaluation	ABC	ABC	ABC	ABC	ABC	ABC	ABC	ABC	AI Pred ABC	AI Pred ABC
Module	A2C	GP TR	EAC BSBO	GCN DNN RL	GraphSage LSTM PPO	GCN MCTS PPO	GCN MDP RL	RL-Env GCN MDP PPO	GCN FC Layer Explorer	GraphSage Transformer MSCAT Explorer
Solution	Operator	Recipe	Recipe	Recipe	Operator	Operator	Operator	Operator	Recipe	Operator Recipe

A new approach for evaluation is based on an AI model, as shown in Fig. 1. Specifically, given a circuit G and a recipe O, we no longer need to perform logic optimization, technology mapping, or QoR calculation. Instead, we only need to train an AI model to evaluate the QoR. The AI-based evaluation method takes approximately 11.04 seconds for 500 optimization recipes, making it 282.56× faster than traditional evaluation. Once we achieve significant acceleration, we can explore 282.56× recipes within a given time. And then, with various heuristic probabilistic optimization methods, we have a high likelihood of discovering higher-quality recipes and logic optimization results.

Additionally, to ensure the QoR of each recipe, accurate AI evaluation is required, meaning we need to minimize the $MAPE(QoR, QoR')$ as shown in Fig. 1. Therefore, we have designed the QoR prediction model CrossLO, which achieves higher accuracy compared to existing techniques. If it is not possible to guarantee a small $MAPE$, then the AI evalution model should ensure that the top few solutions with high QoR still rank among the highest. Our experiments show that our AI model can achieve both small $MAPE$ and high rank maintenance rate of the top 10%.

2.3 Related Works

2.3.1 QoR Prediction. A modular fusion network architecture is utilized, which correlates initial circuits and optimization recipes with the final QoR. Recent advances in logic synthesis within the realm of ML research include the introduction of OpenABC-D by Chowdhury et al. [8], a comprehensive and well-annotated dataset that establishes a benchmark for ML-assisted IC synthesis. OpenABC enables the construction of universal learning frameworks, achieving pioneering results in QoR prediction by evaluating existing solutions. Chenghao Yang et al. [47] employed a circuit feature extractor based on three typical GNNs (GCN [21], GraphSage [16], and GIN [44]) and innovatively proposed the use of Transformer networks [39] to extract features from optimization recipes. They adopted a joint learning strategy that combines GraphSage and Transformers, effectively enhancing the predictive performance of unknown circuit optimization recipe pairs. LOSTIN [42] integrates GNNs for graph learning with Long-Short-Term Memory Networks (LSTM) [17] for the optimization of recipe encoding. Based on LOSTIN, GNN-H [43] represents each synthesis process as a fixed-length vector to generate super-node embeddings, predicting QoR trajectories with high precision. Methods in often simplistically merge graph embeddings and recipe embeddings, using a MLP decoder to generate integrated QoR predictions. MTLSO [10] adopts a multi-task learning framework to jointly optimize graph classification and regression tasks. This approach enables the model benefits from shared representations and exploit inter-task dependencies. They primarily focus on enhancing prediction accuracy and have not been fully integrated into actual LO exploration processes.

2.3.2 Logic Optimization Exploration. Logic optimization exploration incorporates a variety of mainstream methods, as summarized in Table 1. However, existing methods are time-consuming, and their performance is fixed by the deterministic nature of the algorithms, leaving little room for further improvement. In the following sections, we will introduce and analyze the existing works one by one. DRiLLS [18] is designed by mapping comprehensive logical optimization problems to the game environment and deploying A2C agents to find the best optimization path. Recipe optimization is achieved with a single adjustment operator. BOiLS [13] introduced the first algorithm to apply modern Bayesian optimization to a logical synthetic operation space. It effectively balances the dynamic process of exploration and exploitation through Gaussian Processes (GP) and Trust-Region Constrained Acquisition Functions. An innovative sequential black-box optimization method [11] is proposed, which utilizes embedded alignment units and proxy models to balance exploration and exploitation to achieve efficient optimization exploration in logical synthesis. Both the BOiLS and black-box are optimized by adjusting the entire recipe. The core idea of GraphML [14], Yang et al. [47], ABC-RL [7], Zhu et al. [51] and EasySO [49] is to utilize GNNs to represent the state and information of circuits and classify them into appropriate actions within the RL framework. This enables the selection of optimization operators and the analysis of rewards at each step. By maximizing the cumulative rewards, these methods achieve the optimal QoR for circuits. Moreover, each method incorporates unique optimization strategies in the optimization process, further enhancing their performance. ReLS [22] effectively makes use of GNN and retrieval database to guide search algorithms. Existing works are limited by the performance ceiling determined by the fixed algorithms. The AiLO framework can flexibly replace the evaluation and exploration components, enabling incremental growth. We design a more powerful AI evaluation model, CrossLO, and integrated it into the AiLO framework. This integration significantly reduces the dependence on time-consuming assessments using traditional EDA tools. The evaluation module can provide QoR for different recipe lengths based on the exploration approach, enabling feedback for single-step, multi-step, or complete recipe solutions.

3 AiLO FRAMEWORK

In this section, we will delve into the construction and operation of the AiLO framework. As shown in Fig. 2, the AiLO framework is composed of two core parts: AI-based metric evaluation and optimization exploration. The AI evaluation module mainly has two key components: QoR prediction and ranking. In the QoR prediction part, we can design a neural network to evaluate optimization metrics. In this work, we introduce CrossLO to achieve a high evaluation accuracy and a good generalization of the prediction of unseen recipes. In addition, it is challenging to accurately evaluate metrics for some evaluation tasks. In such cases, our primary objective is to provide a ranking of the metrics among the various solutions. The QoR ranking module is used to rank the solution candidates. For optimization exploration, there are many effective techniques: such as the heuristic search, Bayesian optimization, and reinforcement learning. In this work, we embed the above techniques into AiLO framework as search engines, use AI evaluation auxiliary logic optimization exploration. We design rich experiments to demonstrate the effectiveness and flexibility of AiLO.

3.1 CrossLO Model

We show the overview of our CrossLO model in Fig. 3, it is composed of three key parts: (1) the GNN encoder for the AIG, which captures its structural features; (2) the recipe encoder, enhanced by a self-attention Transformer, which learns the overall LO process of the recipe; (3) a multi-scale cross-attention Transformer that integrates both graph and recipe encoding to predict QoR across local and global domains for the execution of the optimized recipe.

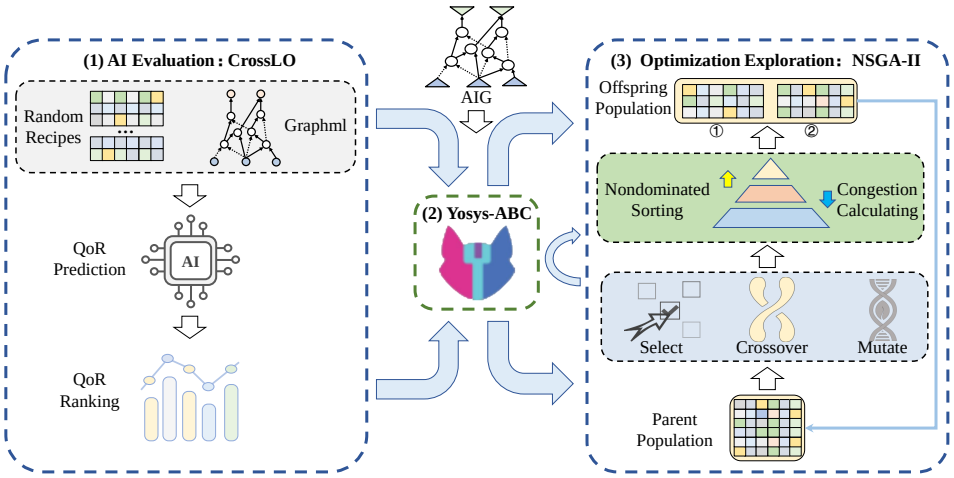


Fig. 2. The Framework of AiLO: (1) The AI evaluation module can quickly evaluate a large number of randomly generated recipes, and submit high-quality recipes to the optimization exploration module; (2) Yosys-ABC tests recipes to generate real QoR information for the optimization exploration module; (3) The optimization exploration module further optimizes and explores the optimal solution according to the guidance of the AI evaluation module

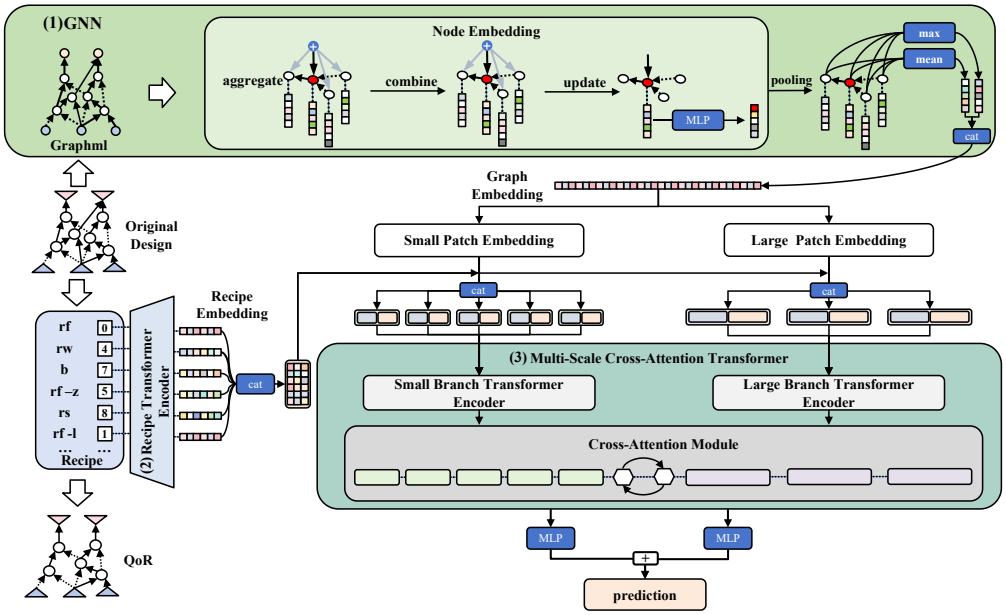


Fig. 3. The Architecture of the CrossLO Model

3.1.1 Graph Encoder. We use GraphSage [16] to obtain graph embeddings $\mathbf{h}_G$. The GraphSage architecture consists of five layers, where each layer performs aggregation and combination processes, iteratively updating node representations by summing the node and edge embeddings.

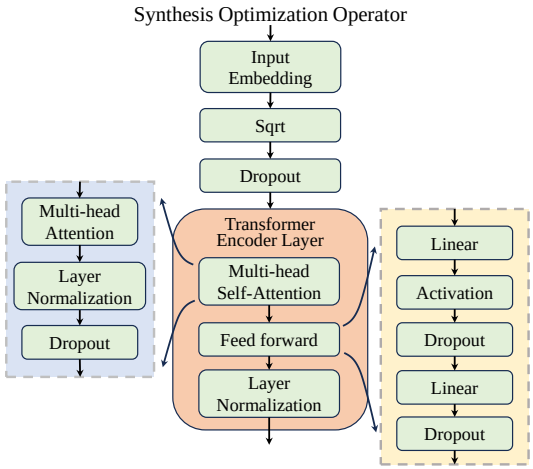


Fig. 4. Logic Optimization Recipe Encoder

Specifically, the node representation $\mathbf{h}_v^{(l)}$ in layer l is updated by aggregating the previous representation $\mathbf{h}_v^{(l-1)}$ of the node v with the embeddings $\mathbf{h}_u^{(l-1)}$ of neighboring nodes $u \in \mathcal{N}(v)$ and $\mathbf{h}_e^{(l-1)}$ of edges $e = (u, v)$ within the neighborhood $\mathcal{N}(v)$ of the node v [19]. After L iterations, the representation of the node $\mathbf{h}_v^{(L)}$ is derived through the aggregation and combination procedures.

$$\mathbf{a}_u^{(l)} = \text{AGGREGATE}(\{\mathbf{h}_u^{(l-1)} : \forall u \in \mathcal{N}(v)\}, \{\mathbf{h}_e^{(l-1)} : e = (v, u)\}) \tag{1}$$

$$\mathbf{h}_v^{(l)} = \text{COMBINE}(\mathbf{h}_v^{(l-1)}, \mathbf{a}_u^{(l)}) \tag{2}$$

where $\mathbf{a}_u^{(l)}$ denotes the aggregation of node and edge embeddings at the l_{th} iteration, $\mathbf{h}_v^{(l)}$ represents the updated node embedding, $\mathbf{h}_u^{(l)}$ is the embedding of the neighboring node $u \in \mathcal{N}(v)$, and $\mathbf{h}_e^{(l)}$ is the edge embedding between nodes u and v. The update is performed using an MLP, followed by a ReLU activation function. The final node embeddings $\mathbf{h}_v^{(L)}$ at layer L are then used to derive the graph embedding $\mathbf{h}_G$. A graph pooling operation is applied, where each node embedding $\mathbf{h}_v^{(L)}$ undergoes averaging and maximum aggregation during the last message passing iteration L. The resulting vectors are concatenated to produce the final graph embedding $\mathbf{h}_G$:

$$\mathbf{h}_G = \text{cat}\left(\frac{1}{N}\sum_{v \in \mathcal{V}} \mathbf{h}_v^{(L)}, \max_{v \in \mathcal{V}} \mathbf{h}_v^{(L)}\right) \tag{3}$$

3.1.2 Recipe Encoder. The encoder architecture as shown in Fig. 4, extracts feature representations from optimized recipes. Each operation is mapped to an embedding vector $\mathbf{e}_i$ through the embedding layer, followed by processing through multiple Transformer encoder layers. Within each layer, the embedding vectors undergo the multi-head self-attention mechanism, described in Equation (4), to capture inter-dependencies and infer the global structure of the recipe [28]. This mechanism preserves long-distance interactions, maintaining the global context of the data.

The multi-head self-attention mechanism is briefly introduced here. For detailed explanation, see the papers [39?].

$$\text{MultiHead}(\mathbf{Q}, \mathbf{K}, \mathbf{V}) = \text{Concat}\left(\text{head}_1, \ldots, \text{head}_h\right) W^O \tag{4}$$

The output is processed through layer normalization to standardize the distribution of the embedded vectors:

$$\mathbf{LN}\left(\mathbf{A}_i\right) = \gamma\left(\frac{\mathbf{A}_i - \mu}{\sqrt{\sigma^2 + \epsilon}}\right) + \beta \tag{5}$$

The normalized vectors are passed through a feedforward network for further feature extraction, as described in:

$$\mathbf{F}_i = \max\left(0, \mathbf{LN}\left(\mathbf{A}_i\right) W_1 + b_1\right) W_2 + b_2 \tag{6}$$

This architecture enables the encoder to effectively capture complex relationships within the optimization recipe, providing high-quality feature representations $\mathbf{h}_r^n$ for logic synthesis optimization, where n denotes the length of the optimization recipe.

3.1.3 Multi-Scale Cross-Attention Transformer. Inspired by the works of CrossViT [6] and Luis H. M. Torres et al. [38], we propose an innovative cross-attention mechanism that leverages a feature fusion strategy to combine the one-dimensional features from two branches using multi-scale representations in one-dimensional graph embeddings. For any graph structure, there exists a reconstruction operator that satisfies a specified error bound, thereby ensuring the completeness of the graph structure decomposition and providing the necessary theoretical foundation for multi scale decomposition [4]. Moreover, the lower-bound theorem of local-global information flow reveals the information gain across attention mechanisms, thereby providing theoretical support for the effectiveness of multiscale Transformers in terms of information flow and pattern learning capabilities [26]. Collectively, these theorems validate the feasibility and efficacy of the multi-scale cross-attention Transformer in handling complex structures. The multi-scale Transformer architecture demonstrates more outstanding deep learning representation capabilities compared to the single-scale architecture. The research results of CrossViT [6] and Liu et al. [26] have been fully proved and experimentally verified, clearly revealing the significant advantages of multi-scale cross-learning. Fig. 5 shows that the proposed multi-scale Transformer encoder serves as a convergence module $\mathcal{G}$, integrating graph and recipe embeddings. The aim is to propagate the graph embedding $\mathbf{h}_G$ and the optimized recipe embedding $\mathbf{h}_r^n$ across two independent branches at different scales.

Both graph and recipe embeddings are treated as recipes of one-dimensional (1D) embedding tokens. The model functions as a graph attribute predictor at varying scales, with the graph embedding h_G mapped onto a recipe of patches x in terms of 1D visual features. By dividing the 1D feature vectors into $N = (\lfloor h_G/P \rfloor)^2$ patches of size P, each patch denoted as x_p, the optimization operators are encoded into corresponding patches r_p. Both patch embeddings are concatenated and projected into the Transformer dimension (D).

$$\mathbf{x} = [x_p^1, x_p^2, \ldots, x_p^N, r_p^1, r_p^2, \ldots, r_p^n] \tag{7}$$

$$\mathbf{p}_{\text{patch}} = \mathbf{x}\mathbf{W} \tag{8}$$

where $\mathbf{x}_{cls}$ is appended to the patch embeddings as an aggregate indicator for the classification output vector. The Transformer also incorporates a positional embedding $\mathbf{p}_{pos}$ to locate elements within the input recipe and learn positional information.

The Transformer block comprises a multi-head self-attention (MSA) layer followed by a feed-forward (FF) network. The MSA operation employs a set of queries $\mathbf{Q}$, keys $\mathbf{K}$, and values $\mathbf{V}$, derived from the input patch embeddings, each of dimension D. As shown in Equation (9), the attention computation involves the dot product of each query in $\mathbf{Q}$ with keys in $\mathbf{K}$, applying the softmax function, and computing the attention weights for each value in $\mathbf{V}$.

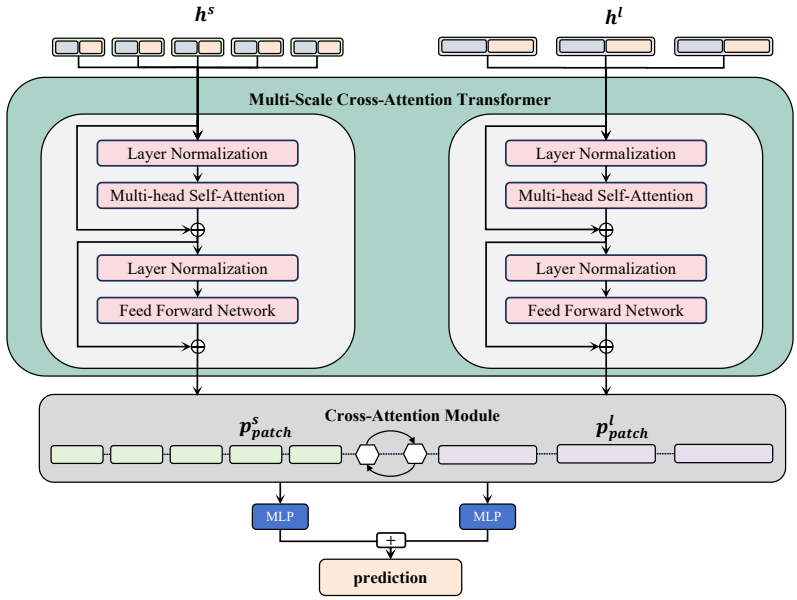


Fig. 5. Multi-Scale Cross-Attention Transformer

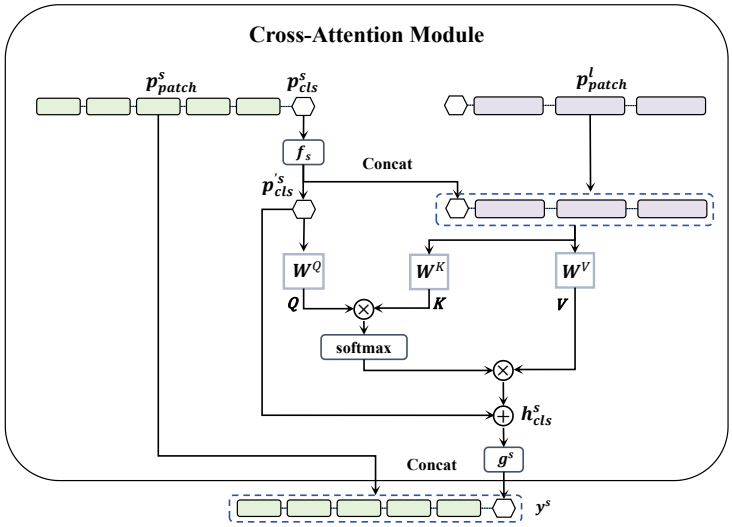


Fig. 6. Cross Attention Module

$$\text{Attention}(\mathbf{Q}, \mathbf{K}, \mathbf{V}) = \text{softmax}\left(\frac{\mathbf{Q}\mathbf{K}^T}{\sqrt{D}}\right)\mathbf{V} \tag{9}$$

$$\text{Attention}_{\text{Multi-Head}}(Q, K, V) = \text{CONCAT}(\text{head}_1, \ldots, \text{head}_H)\, W \tag{10}$$

$$\mathbf{head_j} = \text{Attention}\left(QW_j^Q, KW_j^K, VW_j^V\right) \tag{11}$$

The matrices $(\mathbf{W}_j^Q, \mathbf{W}_j^K, \mathbf{W}_j^V)$ represent the linear projections of the queries, keys, and values.

Within the Transformer block, the feed-forward network consists of a two-layer MLP, followed by a GELU activation function after the first linear layer. Layer normalization and residual connections are applied between the MSA and the feed-forward networks. The deep representations $\mathbf{h}_T$ propagated through the attention layer are as follows:

$$\mathbf{h}^{(0)} = [\mathbf{p}_{cls}] \left[\mathbf{p}_{\text{patch}}\right] + \mathbf{p}_{\text{pos}} \tag{12}$$

$$h_T^{l^*} = h_T^{l-1} + MSA\left(LN\left(h_T^{l-1}\right)\right) \tag{13}$$

$$h_T^{l} = h_T^{l^*} + FF\left(LN\left(h_T^{l^*}\right)\right) \tag{14}$$

where $l = 1, 2, \ldots, L$, $\mathbf{h}^{(l)}$ represents the hierarchical representation at depth l, $\mathbf{p}_{cls}$ is the classification token, $\mathbf{p}_{\text{patch}}$ is the patch embedding token, and $\mathbf{p}_{pos}$ is the positional embedding token.

The multi-scale Transformer utilizes varying patch sizes to generate multi-scale representations of graph embeddings for QoR prediction. The goal is to merge fine-grained and coarse-grained patch information by propagating 1D graph embeddings through two independent branches at different scales. The model consists of L Transformer blocks, each divided into:

- Small-Branch Transformer Encoder: Accepts smaller patch sizes to learn complementary representations, focusing on local connections within embeddings. By retaining more detailed dimensions (e.g., higher feature dimensions or shorter recipe lengths), the model is forced to capture local patterns (such as gate-level connections and the local effects of partial optimization operators on subgraphs).
- Large-Branch Transformer Encoder: Accepts larger patch sizes to learn broader representations, capturing global structure and providing a wider context for better generalization across diverse graph attributes. By reducing the dimensionality through pooling to compress the image and recipe length, the attention mechanism is more inclined to focus on global dependencies (such as the connections between subgraphs relative to the small-branch and the impact of longer recipe on larger subgraphs).

Embedding information is exchanged between the two branches, merging patch embeddings with the classification tokens of both branches, leveraging multi-scale representations of 1D graph embeddings through effective feature fusion.The cross-attention module facilitates the exchange of patch embeddings between the small and large branches. Specifically, the classification token of the small branch interacts with the patch embeddings of the large branch, and vice versa. This process is formalized as:

$$\mathbf{p}'_{cls} = [\mathbf{f}(\mathbf{p}_{cls}) \parallel \mathbf{p}_{patch}] \tag{15}$$

where $\mathbf{p}'_{cls}$ represents the input embedding for cross-attention, $\mathbf{f}(\mathbf{p}_{cls})$ is the transformation of the small branch classification token, and $\mathbf{p}$patch represents the patch tokens. The module computes the cross-attention between the interaction token $\mathbf{p}'$ and $\mathbf{p}_{cls}$, consolidating the compacted patch information summarized by the classification token in the smaller branch. Analogous to the Multi-Head Self-Attention (MSA) mechanism, different heads are generated through linear projections of queries $\mathbf{Q}$, keys $\mathbf{K}$, and values $\mathbf{V}$, which are mapped to distinct embedding spaces. Multi-Head Cross-Attention (MCA) is preceded by layer normalization and residual connections. The output of the cross-attention mechanism is computed as follows:

$$\mathbf{h}_{cls} = \mathbf{f}(\mathbf{p}_{cls}) + \mathcal{M}(\mathcal{L}([\mathbf{f}(\mathbf{p}_{cls}) \parallel \mathbf{p}_{patch}])) \tag{16}$$

$$\mathbf{y} = [\mathbf{g}(\mathbf{h}_{cls}) \parallel \mathbf{p}_{patch}] \tag{17}$$

Here, $\mathbf{h}_{cls}$ represents the updated classification token, and $\mathbf{y}$ denotes the output tokens of the minor branch. The functions $\mathbf{f}$ and $\mathbf{g}$ are projection functions, while $\mathbf{p}_{cls}$ corresponds to the original

classification token of the minor branch. Additionally, $\mathbf{p}_{patch}$ refers to the patch tokens of both the major and minor branches. Following the cross-attention mechanism, two Multi-Layer MLP heads employ the updated classification tokens, $\mathbf{p}_{cls}^{s}$ and $\mathbf{p}_{cls}^{l}$, from both branches to predict the attributes of various tokens. The summation of these predictions yields the ultimate design QoR.

3.2 AI Evaluation Module

In the AiLO Framework, the AI evaluation module is responsible for the efficient assessment of a vast array of randomly generated logic optimization recipes. This module employs deep learning models to predict two key performance metrics of the netlist: area and delay, following the execution of optimization recipes in AIG for logic optimization and technology mapping. To quantify these metrics for subsequent optimization processes, we adopt a Gaussian standardization method to ensure comparability across different indicators.

$$I = \frac{x - \mu}{\sigma} \tag{18}$$

x represent the original area and delay values, while μ and σ denote the mean and standard deviation of the area and delay metrics, respectively. Through Gaussian normalization, we transform these two metrics into distributions with zero mean and unit variance, thereby granting them balanced weight in the subsequent QoR calculations.

In pursuit of a balance between the area and delay metrics of the netlist, we employ a linear weighting approach to calculate the QoR metric:

$$I_{\text{QoR}} = e'(p) = w_{\text{area}} \cdot I_{\text{area}} + w_{\text{delay}} \cdot I_{\text{delay}} \tag{19}$$

w_{area} and w_{delay} are the weights assigned to the area and delay metrics, respectively. These weights can be adjusted according to the requirements of the specific application scenario, reflecting the significance of different metrics. Let P denote the set of all possible optimization recipes, e' represent the performance evaluation function, and Q be the set of QoR metrics. The objective of the AI-based evaluation module is to identify the recipe within P that yields the optimal performance, which can be mathematically formulated as:

$$\min_{p \in P} e'(p) \tag{20}$$

$e'(p)$ signifies the performance score associated with the recipe p. Utilizing the AI evaluation module, we select the top-performing 10% of recipes from P based on their performance scores, denoting this subset as $P' \subset P$:

$$P' = \{p \in P \mid e'(p) \text{ is in the top } 10\%\} \tag{21}$$

By ranking all recipes according to their QoR, we can identify the top-performing recipes. We select the P' with the highest QoR as elite recipes. This step ensures that the AiLO framework focuses its resources on further exploring and optimizing the most promising recipes. It has been verified that the AI evaluation module can be described as generating a distribution range for logic optimization recipes, essentially serving as a preliminary screening of optimization recipes. During the iterative process of solution optimization, the surrogate model often needs to be continuously updated to better predict performance metrics, whereas the AI evaluation module provides a curated set of elite recipes from the outset.

3.3 Explore Algorithm and AI Integration

Since the optimal recipe executed by the design directly affects the QoR of the final design, logic synthesis recipe exploration involves finding an optimal optimal recipe in the circuit design to minimize the area area and delay. The QoR is defined as a function that balances area and delay,

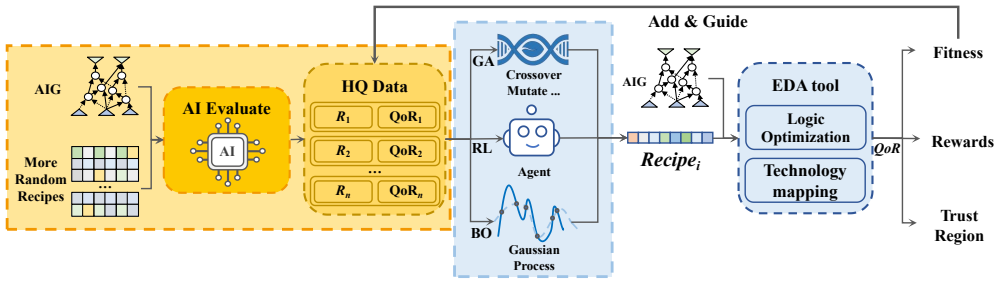


Fig. 7. Explore Algorithm and AI Integration

and is maximized when area and delay of the circuit are minimized relative to a baseline synthesis recipe, Resyn2. The problem can be formulated as that maximizes the QoR for circuit $\mathbb{C}$. This can be expressed as:

$$\max_{\mathbb{C}} \text{QoR} = \max_{\mathbb{C}} \left(2 - \left(\frac{\text{area}(recipe)}{\text{area}(\text{Resyn2})} + \frac{\text{delay}(recipe)}{\text{delay}(\text{Resyn2})}\right)\right) \tag{22}$$

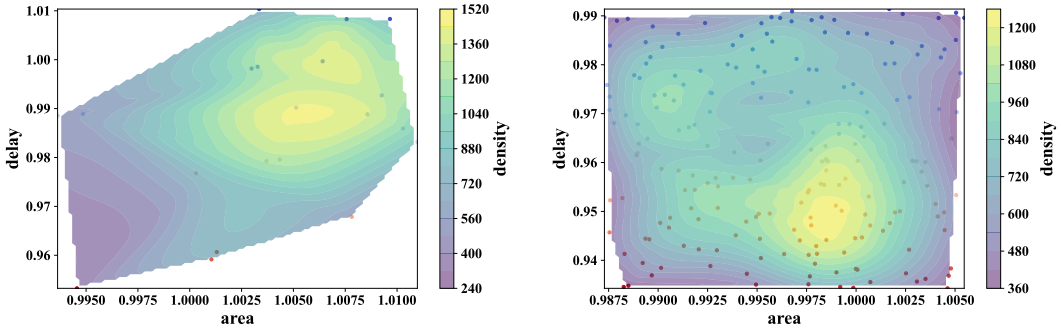
where area($recipe$), delay($recipe$) is the area and delay of the circuit after applying the synthesis recipe. area(Resyn2), delay(Resyn2) is the area and delay of the circuit synthesized by the Resyn2.

Traditional methods use random search and historical data to find solutions. However, the solution space is vast (k^n), making it hard to explore fully. For example, with 13 operators and a recipe length of 10, the solution space is 1.378×10^{11}. Even tens of thousands of iterations cover only a tiny fraction. Moreover, executing optimization recipes and obtaining feedback is time-consuming and resource-intensive. In circuit design, it is impractical to spend much time on thousands of iterations. These methods typically cannot guarantee the quality of recipes at the initial stage, and the distribution of recipes tends to concentrate in areas with poor performance. Fig. 8(a) shows that traditional algorithms start in the upper right corner and try to move towards the lower left for better solutions.

Despite the enhancement in optimization efficiency achieved by current advanced methods, challenges remain in terms of computational costs and sample complexity. It is too difficult to rely solely on the exploration of technological breakthroughs, therefore, we design AI evaluation module to overcome these technical barriers, as shown in Fig. 7. High-quality recipes identified through AI evaluation, as shown in Fig. 8(b), facilitate a hierarchical ranking of optimization recipes, generating a distribution profile that aims to enhance search guidance and pruning strategies, advancing the automation and intelligence of the exploration process in LO.

This paper selects the NSGA-II algorithm due to its remarkable performance in multi-objective optimization. It can effectively identify the Pareto front through non-dominated sorting, maintaining population diversity and solution quality. Although it is somewhat dependent on the initial solutions, it can well assess the impact of the AI evaluation module on logic optimization. Moreover, it is highly computationally efficient, adaptable, and capable of flexibly handling complex optimization problems as well as potential future additions of objectives and constraints.

The overall pseudo-code of the algorithm is shown in Algorithm1. Step 1 uses the AI model e' to rapidly assess a randomly generated set of $10N$ recipes, yielding the corresponding I_{QoR} values. Subsequently, the top 10% of individuals in I_{QoR} are selected to form the initial population P_0. Step 3 to step 18 is the main cycle of algorithm iteration. The iterative process of the algorithm commences by merging the current population P_t with the offspring population Q_t to create a new set R_t. R_t is then subjected to a fast non-dominated sorting procedure, resulting in a sorted



(a) *Previously Random Small Amounts of Data Distribution*

(b) *Generates More Comprehensive, HQ-Data Distribution*

Fig. 8. Comparison of QoR Density Distribution of Different Quantity and Quality Recipes

Algorithm 1 NSGA-II with AI Algorithm

Input: AI model e', recipe length, generations T, the number of individuals in the population N
Output: optimal recipe

$I_{QoR} = e'(10N)$
$P_0 = \{p \in 10N | I_{QoR}$ is in the top 10%$\}$
while $t < T$ **do**
  Get $R_t = P_t \cup Q_t$
  $F =$ fast non dominated sort(R_t)
  Set $P_{t+1} = \emptyset$, $i = 1$
  while $|P_{t+1}| + |F_i| \leq N$ **do**
    crowding distance assignment(F_i)
    $P_{t+1} = P_{t+1} \cup F_i$
    $i = i + 1$
  end while
  Sort$(F_i, \prec_n)$
  $P_{t+1} = P_{t+1} \cup F_i[1 : (N - |P_t + 1|)]$
  $I_{QoR} = e'(10N)$
  $P'_{t+1} = \{p \in 10N | I_{QoR}$ is in the top $(N - |P_{t+1}/2|)$%$\}$
  $Q_{t+1} =$ make new pop$(P_{t+1}/2) \cup P'_{t+1}$
  Set $t = t + 1$
end while

population F. To construct the next-generation population P_{t+1}, the process begins with an empty set and initializes the index $i = 1$. Through iterative steps, individuals from the non-dominated front F_i are added to P_{t+1} based on crowding distance assignment, provided the total number of individuals in P_{t+1}, including those from the current non-dominated front, does not exceed N. If this condition is met, the remaining individuals in F_i are sorted, and a subset is selected to complete the construction of the next-generation population.

In each generation of the iterative process, the AI model e' is again utilized to generate $10N$ new individuals for evaluation. From these newly generated individuals, those ranking in the top $(N - |P_{t+1}|/2)$% of I_{QoR} are selected to form P'_{t+1}. The next-generation offspring population Q_{t+1} is

then created, consisting of half of the individuals from P_{t+1} and half from P'_{t+1}, thereby maintaining population diversity and enhancing exploration capability to prevent premature convergence to local optima. This iterative process continues until the predefined generations T is reached.The selection, mutation, crossover, and non-dominated sorting operations in NSGA-II are elaborated in [5, 9]. Our method retains these components unchanged.

Similar to integrating the AI evaluation module into NSGA-II, we also incorporate AI evaluation modules into reinforcement learning and Bayesian optimization. Based on the open-source codes DRiLLS and BOiLS, we introduce a dual-channel evaluation mechanism: AI-based rapid screening and precise verification. The primary objective is to leverage AI evaluation to filter out high-quality recipes, which are then precisely verified using the traditional EDA tool Yosys-ABC. This enables the training of agents and Gaussian process kernels to identify optimal solutions, learn to maximize rewards, and understand the distribution of high-quality trust regions, thereby enhancing the exploration of superior solutions. For reinforcement learning, because CrossLO can achieve the evaluation of recipes with different lengths, it can predict the QoR of recipe in one or more steps during the running process. This reduces the time cost of each state recognition and action matching. In Section 4.3, we carry out experiments to verify the algorithms.

3.4 Analysis of Deep Learning Applications in Logic Optimization

Deep learning techniques area increasingly applied in the domain of Logic Optimization (LO), particularly for predicting the QoR accuracy. Studies [8, 42, 43, 47] show that these approaches hold potential in this area. Specifically, the significant capabilities of deep learning in evaluating solution quality and assisting with high-dimensional combinatorial search problems in LO can be analyzed from the following perspectives:

- **Solution Quality Assessment**: Deep learning models can evaluate the quality of recipes by learning patterns from historical data. They identify features of high-quality recipes and predict QoR for new ones, aiding in choosing better optimization strategies.
- **Assisting High-Dimensional Combinatorial Search**: ML models can significantly improve search efficiency. They can predict which search paths are more likely to yield high-quality solutions, thereby reducing unnecessary searches and computations.
- **Improvement of Optimization Algorithm Performance**: Optimization algorithms that incorporate AI evaluation outperform their original counterparts in terms of search time and stability. ML models can be used to predict and assess the quality of recipes, while traditional methods leverage these date to guide the search process, enabling more effective pruning and search path selection.

In the logic optimization exploration, the AiLO framework based on deep learning differs from traditional optimization methods by implementing an innovative selection-guidance strategy, enabling fast evaluation before the optimization process begins. To validate the feasibility and effectiveness of this approach, we designed experiments to confirm that, even with slight deviations in the prediction accuracy of deep learning models, the ranking of the generated recipes can still preserve the rank of the recipes. The detailed experimental design is outlined in Section 4.2.5.

In conclusion, the application of AI evaluation module in LO, particularly in high dimensional combinatorial search problems, demonstrates significant potential in improving search efficiency and reducing computational resource requirements. Through AI evaluation and the selection of optimized recipes, ML techniques offer fresh perspectives and solutions to the LO domain.

4 EXPERIMENTAL RESULTS

4.1 Setup

4.1.1 Environment. This section presents a series of experiments to validate the key contributions of this study. These evaluations provided a comprehensive assessment of the architecture's effectiveness and adaptability, focusing on the quality of prediction results and design space exploration. The experimental environment for the following tasks is as follows: The hardware configuration: CPU (Intel Xeon Gold 5118 @ 2.30GHz), Memory (512GB of RAM), GPU (NVIDIA Tesla V100 with 32 GB VRAM), while the software configuration: Operation System (Ubuntu 20.04.4 LTS), Python (3.9.13), PyTorch (1.13.1), CUDA (12.2), torch_geometry (2.5.3), Yosys-ABC (1.01). All algorithms utilize the academic open-source Yosys-ABC to implement specialized heuristic directives for circuitry, supported by the ASAP7 [45] technology library. The tool's print_stats command is used for technology mapping, resulting in a minimized logic circuit optimized for QoR.

4.1.2 Dataset. Logical optimization operators are all derived from ABC, and there are mainly four kinds of optimization operators and their variants, totaling 13 kinds, as follows:

$$\begin{cases} \textit{balance}, \\ \textit{rewrite}, \textit{rewrite -l}, \textit{rewrite -z}, \textit{rewrite -l -z}, \\ \textit{refactor}, \textit{refactor -l}, \textit{refactor -z}, \textit{refactor -l -z}, \\ \textit{resub}, \textit{resub -l}, \textit{resub -z}, \textit{resub -l -z}. \end{cases}$$

The experiment employs two well-known datasets:

- **EPFL** [1] combinatorial Benchmark Suite, launched in 2015, aims to complement existing benchmarking kits by focusing on native combinational logic benchmarks. This suite encompasses 20 circuits designed to test cases for contemporary logic optimization tools, categorized into arithmetic, random, control, and circuits exceeding ten million gates.
- **Open Core** [8] crafted for ML-guided IC logic synthesis, is a substantial compilation based on 29 open-source circuits, spanning a variety of functionalities including communication, processors, and system controllers, suitable for developing and assessing the performance of ML models in logic synthesis tasks.

4.2 AI Evaluation Model

4.2.1 Dataset Preprocessing. Both datasets share a set of 1500 recipes with a Gaussian distribution of lengths averaging 10, capped at a maximum of 20. Of these recipes, 1000 recipes are allocated (70% for training and 30% for validation), while the remaining 500 are utilized to evaluate the generalization of the final model to unseen recipes. Fig. 9 offers an intuitive understanding of the flowchart, facilitating the seamless generation and preprocessing of required data.

(1) **Register Transfer Level (RTL) Synthesis**: Yosys [41] translates RTL descriptions into gate-level netlists, forming the basis for subsequent logic optimization and synthesis.
(2) **Logic Optimization**: Yosys and ABC [3] tools are used for logic synthesis, enhancing circuit performance and reducing resource consumption. Recipes are generated using random strategies, including fixed-length and Gaussian-distributed random-length approaches.
(3) **Technology Mapping and Area Delay Assessment**: ASAP7 [45] library collects area and delay parameters for the technologically mapped AIG.
(4) **Graph Processing**: The AIG is converted to graphml format using Logic Factory [31] and processed with networkx [15] for analysis and optimization.

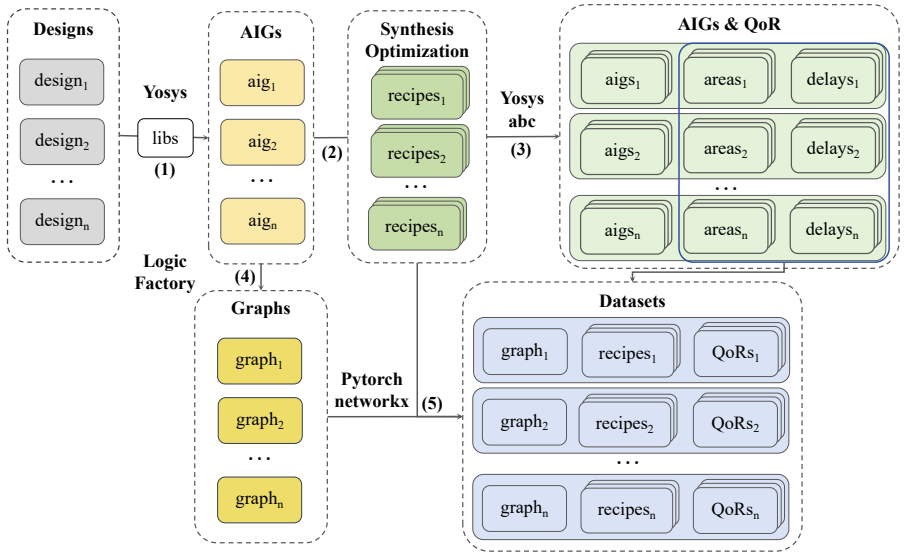


Fig. 9. Data Preprocess Flowchart

Table 2. Training, Validation and Test splits

Split	Open Core	EPFL
Train	i2c, spi, des3_area, ss_pcm, usb_phy, sasc, wb_dma, simple_spi, dynamic_node	bar, max, sin, i2c, cavlc, ctrl, int2float, priority
Valid	aes, pci, ac97_ctrl, mem_ctrl, tv80, fpu	router, sqrt, square, arbiter, adder
Test	wb_conmax, tinyRocket, aes_xcrypt, aes_secworks, jpeg, bp_be, ethernet, vga_lcd, picosoc, dft, idft, fir, iir, sha256	div, log2, multiplier, mem_ctrl, voter, hyp

(5) **Data Alignment and Generation**: PyTorch [32] and PyTorch Geometric [12] are used for aligning and generating data for deep neural network learning, including graph structure, logical features, and circuit characteristics.

The dataset partitioning, as detailed in Table 2, adheres to machine learning benchmarking standards by incorporating both Open Core and EPFL benchmark suites, exhibiting three critical characteristics: (1) a scale-aware design where training sets focus on medium/small-scale circuits (e.g., i2c, spi) to optimize computational efficiency; (2) generalization validation through test sets that include large-scale circuits (e.g., tinyRocket, aes_xcrypt) specifically for evaluating cross-complexity transferability; and (3) distribution equilibrium, where validation sets (e.g., aes, pci) maintain zero overlap with test sets to guarantee unbiased evaluation.

4.2.2 Hyperparameter Setup. For the graph embedding part, an architecture utilizing a five-layer GraphSage is employed for the AIG encoder, with a hidden embedding size of 128. This architecture combines mean pooling and max pooling to generate individual AIG embeddings. For the synthesis recipe embedding part, a four-layer self-attention Transformer is implemented, each layer having a hidden size of 128. In the multi-scale cross-attention Transformer, patch embeddings are set to sizes of 64 and 256, respectively. Two MLPs are used to map the recipe embeddings to match the

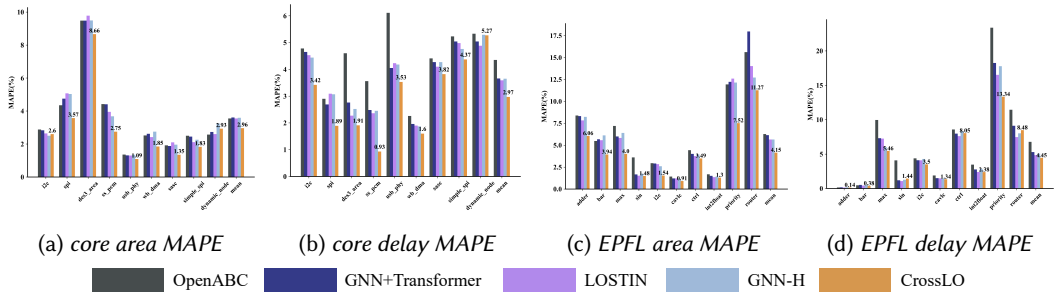


Fig. 10. "Seen IC, Unseen Recipes" Comparative Results of MAPE

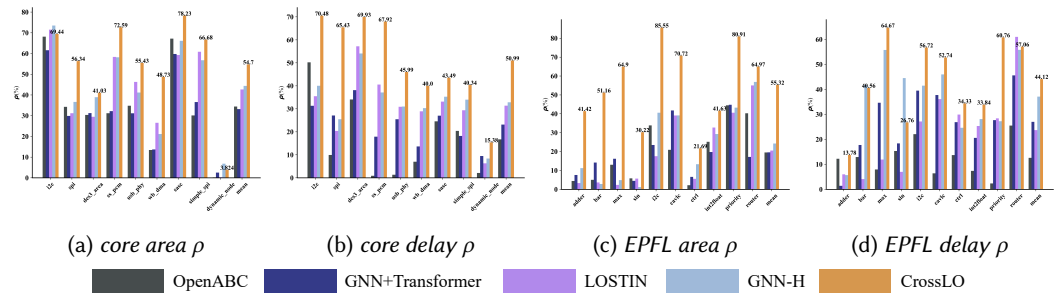


Fig. 11. "Seen IC, Unseen Recipes" Comparative Results of Spearman Rank Correlation Coefficient

patch size and alignment before being input into the cross-attention Transformer. The learning rate is set to 0.0001, the batch size is set to 32, and "Adam" is set as the optimizer; the L1Loss is used for the loss function.

This model employs two evaluation metrics: the Mean Absolute Percentage Error (MAPE) and Spearman rank correlation coefficient [27]:

$$\text{MAPE} = \frac{100\%}{n} \sum_{i=1}^{n} \left| \frac{\hat{y}_i - y_i}{y_i} \right| \tag{23}$$

$$\rho = 1 - \frac{6 \sum d_i^2}{n\left(n^2 - 1\right)} \tag{24}$$

where n represents the total number of evaluation data points, $\hat{y}_i$ denotes the predicted values, and y_i indicates the actual values, d_i is the difference in rank between the predicted and actual values.

4.2.3 *Ablation Study.*

(1) Seen IC, Unseen Recipes. As shown in Table 3, the prediction accuracy of the CrossLO model for the two indicators of area and delay in the two data sets increases by 42.52% on average, and the Spearman rank correlation coefficient increases by 176.66%. Compared with the method of GNN+Transformer, which does not employ the multi-scale cross-attention Transformer, the prediction accuracy is improved by an average of 28.02% across the two datasets, while the Spearman rank correlation coefficient is enhanced by an average of 107.69%. Fig. 10

Table 3. Comparison of QoR Prediction Results under Different Scenarios

Scenario	Methods	Open Core				EPFL			
		area		delay		area		delay	
		MAPE	ρ	MAPE	ρ	MAPE	ρ	MAPE	ρ
Seen IC Unseen Recipes	**OpenABC** [8] (Baseline)	3.55%	34.35%	4.35%	16.25%	6.27%	19.52%	6.78%	12.60%
	LOSTIN [42]	3.55%	42.64%	3.59%	31.36%	5.65%	20.55%	4.85%	23.70%
	GNN-H [43]	3.59%	44.35%	3.65%	32.85%	5.65%	24.25%	5.07%	37.11%
	GNN+Transformer [46]	3.61%	33.17%	3.66%	23.14%	6.15%	19.60%	5.29%	27.02%
	CrossLO	**2.96%**	**54.70%**	**2.97%**	**50.99%**	**4.15%**	**55.32%**	**4.45%**	**44.12%**
	CrossLO vs Baseline	**20.14%**	**59.25%**	**46.45%**	**213.80%**	**51.10%**	**183.37%**	**52.38%**	**250.20%**
Unseen IC Seen Recipes	**OpenABC** [8] (Baseline)	4.22%	19.74%	9.96%	19.28%	5.47%	11.77%	2.22%	18.82%
	LOSTIN [42]	3.70%	2.71%	8.75%	14.07%	**3.18%**	3.45%	1.76%	3.58%
	GNN-H [43]	**3.37%**	35.43%	**8.21%**	**37.22%**	3.30%	23.06%	**1.67%**	**29.93%**
	GNN+Transformer [46]	4.06%	10.43%	8.88%	24.43%	3.79%	13.43%	1.70%	23.08%
	CrossLO	3.47%	**41.05%**	8.50%	30.69%	4.10%	**27.49%**	1.94%	26.86%
	CrossLO vs Baseline	**21.63%**	**107.97%**	**17.21%**	**59.18%**	**33.46%**	**133.63%**	**14.78%**	**42.76%**
Unseen IC and Recipes	**OpenABC** [8] (Baseline)	4.24%	20.58%	10.04%	13.86%	4.50%	14.44%	1.96%	17.22%
	LOSTIN [42]	3.65%	5.04%	8.77%	12.50%	3.52%	4.93%	1.91%	14.62%
	GNN-H [43]	**3.31%**	34.64%	8.50%	32.36%	**3.39%**	20.70%	1.89%	21.07%
	GNN+Transformer [46]	4.07%	10.64%	9.35%	14.47%	3.89%	8.19%	1.93%	13.13%
	CrossLO	3.41%	**39.77%**	**8.00%**	**38.16%**	3.86%	**30.96%**	**1.87%**	**21.94%**
	CrossLO vs Baseline	**24.32%**	**93.24%**	**25.61%**	**175.30%**	**16.69%**	**114.47%**	**5.19%**	**27.42%**

Table 4. The Information Related to the Best Test Loss Obtained during the Training of Various Models

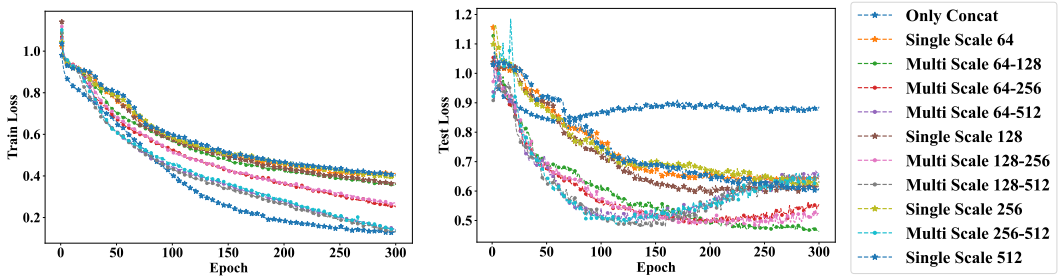
Model	Epoch	Time(h)	MAE Loss	Model	Epoch	Time(h)	MAE Loss
Only Concat	56	0.234	0.838	Multi Scale 64-128	293	5.241	0.463
Single Scale 64	212	2.380	0.635	Multi Scale 64-256	188	3.017	0.474
Single Scale 128	201	2.310	0.593	Multi Scale 64-512	133	2.132	0.489
Single Scale 256	299	2.748	0.619	Multi Scale 128-256	245	4.261	0.476
Single Scale 512	288	2.637	0.597	Multi Scale 128-512	159	2.836	0.481
				Multi Scale 256-512	114	1.810	0.487

and Fig. 11 present detailed comparisons of the prediction performance of different methods in terms of MAPE and Spearman correlation coefficients.

(2) Unseen IC, Seen Recipes. CrossLO achieves a 21.63% improvement in area prediction in the Open Core dataset and a 33. 46% improvement in the EPFL dataset compared to the baseline OpenABC model. For delay prediction, CrossLO improves by 17. 21% and 14. 78% in the respective datasets.

(3) Unseen IC, Unseen Recipes. CrossLO maintains its superiority, achieving a 24.32% improvement in area prediction in the Open Core dataset and a 16. 69% improvement in the EPFL dataset. For delay prediction, the improvements are 25.61% and 5.19%.

Fig. 12 and Table 4 present the training loss curves and time information for models with various configurations, including models with only concatenated graph and recipe embeddings, single-scale and multi-scale models, as well as models with different hyper parameters. By comparing the training and testing loss curves of different models, as well as the optimal testing loss data, the effectiveness of the multi-scale approach for this task is clearly demonstrated. Relative to the



(a) *Train Loss* (b) *Test Loss* (c) *Label*

Fig. 12. Comparison of Loss of Various Models. It includes simple concatenation of Graph embedding and Recipe embedding, single scale and multi scale.

single-scale model and the model with only concatenation, the multi-scale model achieves lower loss values during both training and testing, exhibiting superior generalization ability and training efficiency. This indicates that the multi-scale approach can more effectively capture feature information at different scales, thereby enhancing the model understanding and predictive capabilities.

In summary, these results demonstrate CrossLO strong generalization ability, which is attributed to its innovative multi-scale cross-attention Transformer that enables deeper understanding and capture of complex features and patterns in IC designs. However, CrossLO under performs GNN-H and LOSTIN in some specific metrics. For instance, in the scenario "Seen Recipes, Unseen IC" , GNN-H and LOSTIN achieve delay prediction accuracies 8.21% and 8.75% higher than CrossLO on the Open Core dataset. This may be due to the significant impact of the IC graph structure on CrossLO performance. GNN-H and LOSTIN might have certain advantages in delay prediction for specific IC types due to their unique graph neural network architectures or training strategies. This finding offers valuable insights for our future research. We plan to further investigate CrossLO architecture and training algorithms to achieve comprehensive performance improvements and maintain its leading position in logic optimization QoR prediction.

4.2.4 Runtime comparison. In terms of efficiency, CrossLO has also achieved remarkable results. Table 5 shows a comparison of evaluation 500 recipes time between Yosys-ABC and deep learning neural networks in part of the designs. While traditional EDA tools take an average of 51.99 minutes to process 500 recipes, CrossLO reduces this time to 11.04 seconds, a time of 282.56× faster. This increase in efficiency reduces the waiting time, and provides the possibility for large-scale data processing. Therefore, CrossLO has significant potential for QoR prediction in deep learning neural networks. However, compared with existing methods such as OpenABC, GNN+Transformer, LOSTIN, and GNN-H, CrossLO exhibits certain time-to-consumption differences. These methods, which employ simpler graph encoding and recipe encoding through addition or concatenation, may be faster. In contrast, the multi-scale cross-attention Transformer in CrossLO, though computationally more complex, enables deeper exploration and fusion of IC graph structure and recipe information relationships, significantly boosting prediction performance. Thus, a modest trade-off in computational efficiency for CrossLO is justified and it shows great potential in QoR prediction among deep learning neural networks.

4.2.5 Ranking comparison. The primary objective of this part of the experiment is to assess the effectiveness of the evaluation method in ranking the quality of optimized recipe solutions in

Table 5. Evaluation 500 Recipes Time for Different Methods: PI, PO, AND are the size of circuit design specifications. Yosys-ABC evaluation time unit is minute, and other AI methods evaluation time unit is second.

Designs	PI	PO	AND	Yosys-ABC(m)	OpenABC [8]	LOSTIN [42]	GNN-H [43]	GNN+Transformer [46]	CrossLO
ctrl	7	26	174	6.56	0.55	0.52	0.54	0.65	2.23
router	60	30	257	6.70	0.72	0.86	0.54	0.48	2.48
int2float	11	7	260	6.62	0.57	0.65	0.69	0.47	2.40
cavlc	10	11	693	7.04	0.71	0.62	0.61	0.59	2.17
priority	128	8	978	7.10	0.75	0.81	0.66	0.67	2.67
adder	256	129	1020	7.07	4.45	3.89	6.44	3.34	3.38
i2c	147	142	1342	7.42	0.86	0.97	0.75	0.84	2.90
max	512	130	2865	9.49	1.46	1.15	1.10	1.35	4.13
bar	135	128	3336	8.47	1.70	1.39	1.18	1.47	4.50
sum	24	25	5416	16.51	1.84	1.57	1.51	2.01	4.09
arbiter	256	129	11839	27.53	3.43	2.47	2.74	4.05	6.23
voter	1001	1	13758	21.90	3.94	3.11	3.12	4.48	6.71
square	64	128	18484	34.41	5.29	3.80	3.61	5.75	8.26
sqrt	128	64	24618	56.10	6.00	4.41	4.47	7.24	11.79
multiplier	128	128	27062	47.25	6.64	5.12	4.85	8.13	11.34
log2	32	32	32060	71.08	7.88	6.26	6.18	9.72	12.34
mem_ctrl	1204	1231	46836	52.40	11.71	8.28	8.51	14.02	19.61
div	128	128	57247	55.51	13.70	10.32	10.12	16.83	33.46
hyp	256	128	214335	538.67	51.35	35.96	35.94	57.39	69.07
Mean	236	137	24346	51.99	6.50	4.85	4.92	7.34	11.04

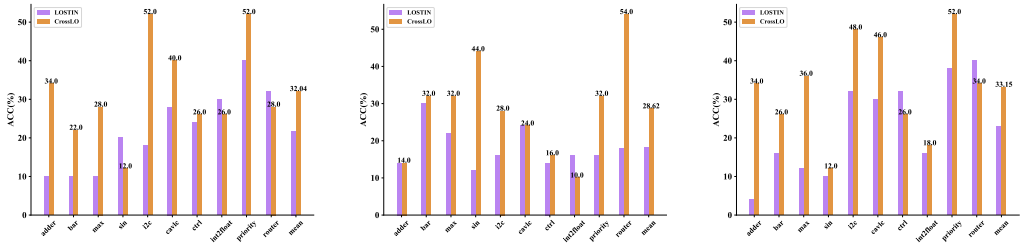
logic comprehensive optimization, and to explore its feasibility in evaluating the rank of optimized recipes under conditions of similar prediction accuracy.

The training and test datasets used in this experiment are derived from the EPFL QoR prediction dataset. We choose the LOSTIN method as the benchmark for comparison because it has high prediction accuracy but relatively low Spearman rank correlation coefficient. The focus is on evaluating the ranking of 500 recipes without prior exposure to the recipes and evaluating the accuracy. The accuracy metric acc is defined as:

$$\text{acc} = \frac{|S_{\text{true}} \cap S_{\text{pred}}|}{N} \tag{25}$$

where: S_{true} is the set of true top 10% recipes, S_{pred} is the set of predicted top 10% recipes, N is the total number of recipes (in this case, $N = 500$), $|\cdot|$ denotes the cardinality of a set. This accuracy is used to verify the effectiveness and feasibility of the AI evaluation. The experimental process included training LOSTIN and CrossLO models to predict area and delay. The area achieved about 6% MAPE, and Spearman correlation coefficients are 20.15% and 40.63%, respectively. Delay predicted that MAPE is about 5%, and Spearman's correlation coefficients are 22.94% and 37.80%, respectively. We evaluate recipes using these two methods on five random samples with similar prediction accuracy, determine their QoR order, and predict the average accuracy of the top 10% QoR recipe relative to the true top 10%.

Fig. 13 visualization results show that demonstrate that under the premise of comparable predictive accuracy between the two models, the CrossLO approach outperforms in the accuracy of ranking recipes for area, delay, and QoR metrics. Specifically, the ranking accuracy rates for CrossLO on these metrics are 32.04%, 28.62%, and 33.15%, respectively, which represent a significant improvement over the LOSTIN rates of 21.80%, 18.14%, and 23.06%. When CrossLO is trained to its optimal state, its accuracy in ranking the top 10% of optimization recipes for QoR reaches 40.46%. These findings confirm the efficacy of the evaluation method in sorting the quality of solutions in logic synthesis optimization and demonstrate that the evaluation method can maintain the ranking of optimization recipes even when there are biases in predictive accuracy.



(a) *area accuracy* (b) *delay accuracy* (c) *QoR accuracy*

Fig. 13. Evaluation of Top 10% Excellent Recipe Accuracy

4.3 Logic Optimization Exploration

4.3.1 Configuration. This experiment evaluates the AiLO method, comparing its performance against prevalent methodologies. Each randomized seed experiment is repeated 5 times to ensure the reliability of the results. It consists of three primary modes:

- **Limit 100 recipes:** A comparison of exploration effects under the Yosys-ABC evaluation with 100 recipes, assessing the logic optimization capabilities of the AiLO model with minimal reliance on traditional EDA tools.
- **Limit 5 hours:** An evaluation of optimization effects within a 5-hour exploration window to gauge the logic optimization efficiency of AiLO model.
- **Different AI evaluation modules in AiLO:** An assessment of the impact of different AI evaluation modules on AiLO by comparing exploration outcomes under the same constraint of 100 recipes.

We randomly select 9 circuits in EPFL benchmark suite, which have different gate sizes, with gate spans ranging from 1,000 nodes to more than 200,000 nodes. Resyn2 is chosen as the benchmark recipe, with the recipe length set to match Resyn2 ten optimization operators. During the exploration process, the optimal recipe from all tested recipes is recorded, and its QoR is considered the final outcome of the experiment. In addition, the training set used by CrossLO in the AI evaluation module is consistent with the EPFL dataset in the QoR prediction experiment. When testing the impact of different AI evaluation modules on the AiLO framework, we choose to combine them with the heuristic algorithm NSGA-II. Because heuristic NSGA-II has less search disturbance than RL and Bayesian optimization, it reduces the impact of search algorithms on logic optimization exploration results. Additionally, NSGA-II relies more on high-quality solutions during iteration, better demonstrating the effectiveness of AI evaluation modules.

The details of the mainstream methodologies are as follows:

- **DRiLLS** [18]: We conduct DRiLLS, a classic reinforcement learning (RL) method. Given that RL strategies rely on immediate feedback, we extend the Yosys-ABC testing conditions to 1000 iterations and optimize the reward mechanism to align with the QoR objectives.
- **NSGA-II**: NSGA-II offers significant benefits in terms of efficiency, variety, flexibility and proven performance.
- **BOiLS** [13]: The algorithm demonstrated exceptional performance with a reduced number of circuit evaluations. We assess the performance of the AiLO recipe exploration through 20 random searches and 80 kernel-tuning searches.
- **Yang et al.** [47]: The method integrates RL with GINE and LSTM to explore logic optimization as a reference for QoR optimization performance.

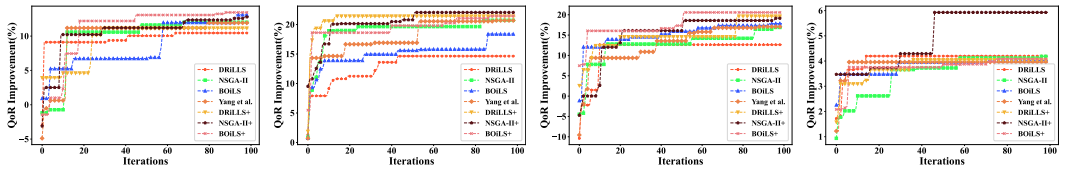
Table 6. Limit 100 Recipes QoR Improvement (in %) Comparison: The main techniques used in the method are marked under the method, and the running time is measured in minutes.★ mark the first place, † mark the second place.

Methods	index	priority	i2c	max	voter	square	multiplier	log2	mem_ctrl	hyp	Mean
Resyn2 (Baseline)	*area*	632.68	837.77	2848.55	13855.36	16210.02	22669.38	26929.54	31503.63	211090.90	-
	delay	2076.20	164.74	2083.68	638.00	2520.69	2677.72	3725.86	1041.40	176561.08	-
DRiLLS [18] **(RL)**	*area*	458.45	798.95	2146.11	11042.59	14836.99	21780.40	26203.02	29104.49	201944.63	-
	delay	929.46	162.24	2117.49	688.83	2539.33	2681.53	3747.92	992.21	176801.92	-
	QoR	82.77	6.15	23.04	12.33	7.73	3.78	2.11	12.34	4.20†	17.16
	time	28.11	31.31	28.81	84.44	141.17	165.68	250.92	262.24	1958.73	327.94
BOiLS [13] **(BO)**	*area*	404.85	796.92	2215.18	10987.76	14822.00	21580.75	26087.98	28074.80	202775.50	-
	delay	885.36	150.80	2070.17	672.20	2541.28	2677.98	3729.65	964.79	176699.62	-
	QoR	93.37†	13.34†	22.88	15.34	7.75	4.79†	3.02	18.24	3.86	20.29
	time	466.86	474.75	476.17	518.39	532.36	556.65	564.23	575.73	632.93	533.12
NSGA-II (HA)	*area*	451.55	812.31	2251.78	11341.17	15785.72	21782.95	26308.24	29308.02	202519.72	-
	delay	903.66	149.83	2096.09	621.80	2265.05	2677.63	3705.36	939.29	176455.23	-
	QoR	85.10	10.74	20.35	20.68	12.76	3.91	2.86	16.77	4.12	19.70
	time	0.99	2.89	2.36	1.98	2.98	4.83	6.95	21.52	46.76	10.14
Yang et al. [47] **(GNN + LSTM + RL)**	*area*	437.24	810.27	2514.86	11801.97	15979.51	21739.22	26255.65	29398.46	203584.98	-
	delay	886.36	151.00	1865.55	611.06	2289.24	2677.75	3711.34	945.48	176760.41	-
	QoR	88.20	11.62	22.18	19.04	10.60	4.10	2.89	15.89	3.44	19.78
	time	34.38	41.25	71.57	205.11	379.69	546.74	647.72	970.09	6258.37	1017.21
DRiLLS+ (CrossLO + RL)	*area*	441.15	816.74	2151.73	11163.47	15737.40	21709.95	26113.06	29364.24	202257.88	-
	delay	907.38	150.50	2072.92	625.27	2253.93	2678.03	3715.13	907.49	176805.45	-
	QoR	86.57	11.15	24.98†	21.42	13.50	4.22	3.32†	19.65†	4.05	20.98
	time	31.17	35.37	37.82	92.53	113.29	147.43	213.09	211.66	1220.87	233.69
BOiLS+ (CrossLO + BO)	*area*	403.23	797.51	2470.05	11705.43	15697	21297	26026.56	28641.33	202157.53	-
	delay	886.36	150.5	1813.46	600.15	2246.67	2678.9	3708.97	921.78	176821.98	-
	QoR	**93.57★**	**13.45★**	**26.26★**	21.45†	**14.04★**	**6.01★**	**3.81★**	**20.57★**	4.08	**22.51★**
	time	459.18	481.38	478.55	517.83	537.08	538.35	572.83	570.83	628.72	531.64
NSGA-II+ (CrossLO + HA)	*area*	399.74	814.16	2208.12	11327.268	15671.89	21672.338	26184.8	28483.8	198301.88	-
	delay	906.42	149.2	2072.92	616.158	2258.97	2677.728	3706.71	938.825	176792.06	-
	QoR	93.16	12.25	23.00	**21.67★**	13.70†	4.40	3.28	19.44	**5.93★**	21.87†
	time	1.02	3.19	2.76	2.32	3.54	5.28	7.53	23.01	52.09	11.19

- **DRiLLS+, NSGA-II+, BOiLS+**: We integrate the AI evaluation module to screen high-quality recipes and guide the original exploration algorithm. We examine the performance of the original algorithm without the integrated AI evaluation module to assess the impact and efficacy of AiLO on algorithmic performance.

4.3.2 Comparison 1: Limitation of 100 recipes. Fig. 14 illustrates the QoR improvement curves for various methods across the optimization exploration process. Table 6 demonstrates that AiLO significantly enhances the efficiency of selecting optimal recipes for IC designs. Even under constrained computational resources, this hybrid algorithm effectively searches for optimal solutions. Algorithms with integrated AI evaluation demonstrate substantial improvements over their original counterparts. Compared to DRiLLS, NSGA-II, and BOiLS without AI evaluation modules, performance enhancements of 22.29%, 11.01%, and 10.96% are achieved, respectively, with an average improvement of 14.75%.

In terms of time efficiency, the integration of CrossLO results in nearly negligible additional evaluation time. For designs with more than 10,000 nodes, BOiLS runs for an average of 9 hours, while NSGA-II+ runs for an average of 10 minutes. In addition, as the number of nodes decreases, AiLO further reduces the run-time ratio of the Yosys-ABC tool. For designs with fewer nodes such as i2c, max and priority, the NSGA+ running time is 1-3 minutes, while the Bayesian optimization of BOiLS still consumes a lot of time, reaching 7-8 hours. NSGA-II+ saves about 200 times to achieve similar or even better QoR improvements than BOiLS. CrossLO supports the evaluation of different recipe lengths, i.e., single or multi-step QoR prediction, which makes the AiLO framework



(a) *i2c* (b) *voter* (c) *mem_ctrl* (d) *hyp*

Fig. 14. QoR Improvement Figures for Some ICs under Limitation of 100 Recipes

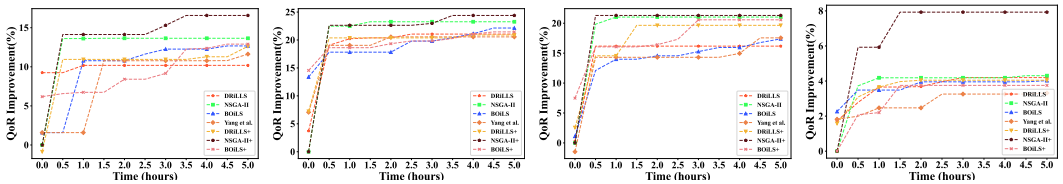
Table 7. Limit 5 Hours QoR Improvement (in %) Comparison: ⋆ mark the first place, † mark the second place.

Methods	index	priority	i2c	max	voter	square	multiplier	log2	mem_ctrl	hyp	Mean
Resyn2 (Baseline)	*area*	632.68	837.77	2848.55	13855.36	16210.02	22669.38	26929.54	31503.63	211090.90	-
	delay	2076.20	164.74	2083.68	638.00	2520.69	2677.72	3725.86	1041.40	176561.08	-
DRiLLS [18] **(RL)**	*area*	411.38	831.5	2135.72	11089.45	15689.98	21618.67	26472.58	28788.16	201944.63	-
	delay	906.42	149.2	2148.33	631.02	2270.29	2679.09	3702.09	962.72	176801.92	-
	QoR	91.32	10.18	21.92	21.06	13.14	4.58	2.33	16.17	4.20	20.54
BOiLS [13] **(BO)**	*area*	414.4	801.51	2135.72	11168.2	16085.49	21741.21	26255.65	28131	202799.67	-
	delay	886.36	151	2148.33	620.47	2257.12	2676.58	3711.34	971.97	176394.19	-
	QoR	91.81	12.67	21.92	22.14	11.22	4.14	2.89	17.37	4.02	20.91
NSGA-II (HA)	*area*	395.96	792.7	2109.08	11397.47	15430.12	21422.14	25773.73	28077.49	201944.07	-
	delay	886.36	151.12	2075.97	602.75	2259.61	2676.63	3710.33	935.63	176611.55	-
	QoR	94.72†	13.65†	26.33	23.26	15.17†	5.54†	4.71†	21.03†	4.30†	23.22†
Yang et al. [47] **(GNN + LSTM + RL)**	*area*	414.15	810.27	2289.44	11391.78	15950.83	21822.00	25942.26	28380.00	203817.07	-
	delay	933.31	151.00	2030.61	620.28	2277.72	2678.61	3715.91	961.83	176793.62	-
	QoR	89.59	11.62	22.17	20.56	11.24	3.70	3.93	17.56	3.31	20.41
DRiLLS+ (CrossLO + RL)	*area*	397.37	801.27	2154.66	11326.33	15676.88	21709.95	26113.06	29364.24	202257.88	-
	delay	883.86	151.12	2075.97	621.17	2243.69	2678.03	3715.13	907.49	176805.45	-
	QoR	94.62	12.62	24.73	20.89	14.28	4.22	3.32	19.65	4.05	22.04
BOiLS+ (CrossLO + BO)	*area*	401.12	792.27	2478.8	11705.43	15697	21660.5	25752.29	28641.33	202878.41	-
	delay	906.42	152.13	1749.07	600.15	2246.67	2678.9	3713.71	921.78	176805.28	-
	QoR	92.94	13.09	29.04†	21.45	14.04	4.41	4.70	20.57	3.75	22.66
NSGA-II+ (CrossLO + HA)	*area*	381.33	805.27	2397.77	11383.77	15421.32	21358.09	25698.05	28421.77	193980.48	-
	delay	869.47	143.84	1737.09	596.19	2259.49	2676.41	3714.45	921.44	176873.47	-
	QoR	**97.85⋆**	**16.57⋆**	**32.46⋆**	**24.39⋆**	**15.23⋆**	**5.83⋆**	**4.88⋆**	**21.30⋆**	**7.93⋆**	**25.13⋆**

more flexible for the integration of RL, reducing the time cost of each state recognition and action matching compared to the original RL. DRiLLS+ reduces time by 40.33% compared to DRiLLS under the constraint of 100 recipes.

During five random exploration trials, instances where algorithms with integrated AI evaluation tools underperformed compared to original algorithms were observed. Notably, while the original DRiLLS achieved an average improvement of 4.20%, the integrated version exhibited a negative optimization of 4.05%. The possible reason for this could be that the ability of the model to evaluate recipes capabilities still needs enhancement, which is evident in the experimental results presented in Section 4.2. Although CrossLO shows an improvement in Spearman Rank Correlation Coefficient for several ICs compares to other models, it is still at a relatively lower level, indicating that there is significant room for improvement in deep learning predictions. Another possibility is that high-quality solutions are not effectively fed back to the reinforcement learning agent, leading to biased selection of optimization operators. Therefore, the interaction between high-quality solutions and the original exploration algorithm may require further investigation to be strengthened.

4.3.3 Comparison 2: Limitation of 5 hours. Fig. 15 and Table 7 show the exploration results of different algorithms under the limited exploration time of five hours. It is clearly observed that the

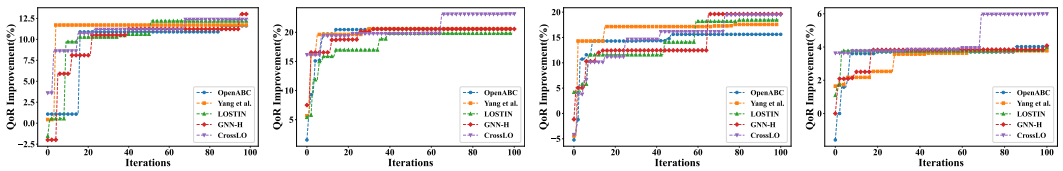


(a) *i2c* (b) *voter* (c) *mem_ctrl* (d) *hyp*

Fig. 15. QoR Improvement Figure for Some ICs in 5 hours

Table 8. NSGA-II and Different AI Evaluation Models Combined with QoR Improvement(in %) Comparison

Models	index	priority	i2c	max	voter	square	multiplier	log2	mem_ctrl	hyp	Mean
NSGA-II (Baseline)	*area*	451.55	812.31	2251.78	11341.17	15785.72	21782.95	26308.24	29308.02	202519.72	-
	delay	903.66	149.83	2096.09	621.80	2265.05	2677.63	3705.36	939.29	176455.23	-
	QoR	85.10	10.74	20.35	20.68	12.76	3.91	2.86	16.77	4.12	19.70
OpenABC [8] **(GCN + FC Layer)**	*area*	411.20	820.48	2441.78	11337.72	15612.21	21819.06	26403.08	29272.77	202376.60	-
	delay	900.77	149.20	1988.89	634.60	2263.68	2677.81	3707.60	932.15	176966.25	-
	QoR	91.62	11.50	18.83	18.70	13.88	3.75	2.45	17.57	3.90	20.24
LOSTIN [42] **(GIN + LSTM)**	*area*	412.86	814.32	2352.37	11567.49	15664.28	21745.50	26313.54	28919.37	202528.76	-
	delay	909.53	149.20	1996.94	614.23	2260.55	2678.41	3706.63	934.67	176639.62	-
	QoR	90.94	12.23	21.58	20.24	13.69	4.05	2.80	18.45	4.01	20.89
GNN-H [43] **(GIN+LSTM+Super-node)**	*area*	413.89	804.50	2463.84	11419.37	15591.28	21765.51	26505.03	28926.18	202340.43	-
	delay	896.41	149.84	1973.78	619.03	2256.25	2677.53	3705.35	919.52	176776.39	-
	QoR	91.41	**13.02**	18.78	20.55	**14.31**	3.99	2.13	**19.88**	4.02	20.90
GNN+Transformer [46] **(GraphSage+Transformer)**	*area*	437.07	812.07	2260.5	11656.11	15640.97	21722.51	26282.01	29233.06	202892.43	-
	delay	868.76	150.5	2075.97	607.78	2258.97	2680.01	3706.05	933.47	176758.42	-
	QoR	89.07	11.71	21.01	20.61	13.89	4.09	2.94	17.57	3.77	20.52
CrossLO (GraphSage+Transformer +MSCAT)	*area*	399.74	814.16	2208.12	11327.27	15671.89	21672.34	26184.80	28483.80	198301.88	-
	delay	906.42	149.20	2072.92	616.16	2258.97	2677.73	3706.71	938.83	176792.06	-
	QoR	**93.16**	12.25	**23.00**	**21.67**	13.70	**4.40**	**3.28**	19.44	**5.93**	**21.87**



(a) *i2c* (b) *voter* (c) *mem_ctrl* (d) *hyp*

Fig. 16. NSGA-II and Different AI Evaluation Models in QoR Exploration under Limitation of 100 Recipes

NSGA-II algorithm combined with AI evaluation has outstanding performance, which is significantly superior to other methods by means of high-quality solution population and fast exploration iteration, and has an improvement of 8.49% compared with NSGA-II without AI evaluation. The QoR improvement curve shows that NSGA-II+ effectively overcomes the performance limitations of traditional NSGA-II, avoiding premature convergence to local optima. Conversely, DRiLLS is hindered by excessive reliance on immediate EDA feedback, and BOiLS is slowed down by kernel inference, both of which are time-consuming and detrimental to efficient logic optimization exploration. This highlights the significant efficiency advantage of NSGA-II+.

4.3.4 Comparison 3: Different AI evaluation modules in AiLO. Table 8 and Fig. 16 show the experimental results of combining NSGA-II with various AI evaluation modules. The average QoR

results indicate that integrating AI evaluation modules into NSGA-II improves performance, proving the effectiveness of the AiLO framework. However, some ICs, like OpenABC and LOSTIN in max, and OpenABC, GNN+Transformer in hyp, show negative optimization compared to NSGA-II without AI evaluation. This suggests unreliable AI models might mislead the search algorithm. Therefore, the AI evaluation component in the AiLO logic optimization framework is replaceable and improves with better AI evaluation models.

In summary, reinforcement learning, heuristic algorithms, and Bayesian optimization algorithms enhanced by AI evaluation modules have demonstrated excellent performance in selecting high-quality recipes while reducing dependence on direct feedback from traditional EDA tools. This enables effective pursuit of optimal solutions within limited computational resources. Moreover, the high flexibility of the AiLO framework means that performance improvements in AI evaluation modules and logic optimization exploration algorithms can directly or indirectly enhance the overall performance of AiLO. New research can also directly utilize this framework to achieve improvements in logic synthesis.

5 CONCLUSION

We develop CrossLO model, significantly enhancing the accuracy of QoR prediction. Experiments confirm that even with a little deviation in prediction accuracy, AI evaluation module can still effectively keep the rank of recipes. Based on this, we design AiLO framework to achieve pruning and exploration. This approach reduces reliance on traditional EDA tools, intelligently guides the exploration path, effectively improves the quality and speed, and realizes AI-enabled LO.

For future work, we are committed to further improving the generalization capabilities and predictive accuracy of the CrossLO model, with the goal of applying it to a broader range of design cases. Additionally, we aim to explore its potential integration with existing EDA tools to achieve a more efficient automated design process. These efforts are expected to inject new vitality into technological advancements in the EDA domain, providing a solid theoretical foundation and practical guidance for both academia and industry.

REFERENCES

[1] Luca Amarú, Pierre-Emmanuel Gaillardon, and Giovanni De Micheli. 2015. The EPFL Combinational Benchmark Suite. In *Proceedings of the 24th International Workshop on Logic & Synthesis (IWLS)*.

[2] Armin Biere. 2007. *The AIGER And-Inverter Graph (AIG) Format Version 20071012*. Technical Report 07/1. Institute for Formal Models and Verification, Johannes Kepler University.

[3] Robert Brayton and Alan Mishchenko. 2010. ABC: An Academic Industrial-strength Verification Tool. In *Computer Aided Verification: 22nd International Conference, CAV 2010, Edinburgh, UK, July 15-19, 2010. Proceedings 22*. Springer, 24–40.

[4] Michael M. Bronstein, Joan Bruna, Yann LeCun, Arthur Szlam, and Pierre Vandergheynst. 2017. Geometric Deep Learning: Going beyond Euclidean data. *IEEE Signal Processing Magazine* 34, 4 (2017), 18–42. https://doi.org/10.1109/MSP.2017.2693418

[5] Maxim Buzdalov and Anatoly Shalyto. 2014. A Provably Asymptotically Fast Version of The Generalized Jensen Algorithm For Non-Dominated Sorting. 8672 (2014), 528–537.

[6] Chun-Fu Richard Chen, Quanfu Fan, and Rameswar Panda. 2021. CrossViT: Cross-Attention Multi-Scale Vision Transformer for Image Classification. In *Proceedings of 2021 IEEE/CVF International Conference on Computer Vision (ICCV)*. 347–356.

[7] Animesh Basak Chowdhury, Marco Romanelli, Benjamin Tan, Ramesh Karri, and Siddharth Garg. 2024. Retrieval-Guided Reinforcement Learning for Boolean Circuit Minimization. In *Proceedings of the 12th International Conference on Learning Representations (ICLR)*.

[8] Animesh Basak Chowdhury, Benjamin Tan, Ramesh Karri, and Siddharth Garg. 2021. OpenABC-D: A Large-Scale Dataset For Machine Learning Guided Integrated Circuit Synthesis. *arXiv preprint arXiv:2110.11292* (2021).

[9] Kalyanmoy Deb, Amrit Pratap, Sameer Agarwal, and TAMT Meyarivan. 2002. A Fast and Elitist Multiobjective Genetic Algorithm: NSGA-II. *IEEE Transactions on evolutionary computation* 6, 2 (2002), 182–197.

[10] Faezeh Faez, Raika Karimi, Yingxue Zhang, Xing Li, Lei Chen, Mingxuan Yuan, and Mahdi Biparva. 2025. MTLSO: A Multi-Task Learning Approach for Logic Synthesis Optimization. In *Proceedings of the 30th Asia and South Pacific Design Automation Conference (ASPDAC)*. New York, NY, USA, 72–78.

[11] Chang Feng, Wenlong Lyu, Zhitang Chen, Junjie Ye, Mingxuan Yuan, and Jianye Hao. 2022. Batch Sequential Black-Box Optimization with Embedding Alignment Cells for Logic Synthesis. In *Proceedings of the 41st IEEE/ACM International Conference on Computer-Aided Design (ICCAD)*. New York, NY, USA, Article 56, 9 pages.

[12] Matthias Fey and Jan Eric Lenssen. 2019. Fast Graph Representation Learning with PyTorch Geometric. *arXiv preprint arXiv:1903.02428* (2019).

[13] Antoine Grosnit, Cedric Malherbe, Rasul Tutunov, Xingchen Wan, Jun Wang, and Haitham Bou Ammar. 2022. BOiLS: Bayesian Optimisation for Logic Synthesis. In *Proceedings of 2022 Design, Automation & Test in Europe Conference & Exhibition (DATE)*. 1193–1196.

[14] Winston Haaswijk, Edo Collins, Benoit Seguin, Mathias Soeken, Frédéric Kaplan, Sabine Süsstrunk, and Giovanni De Micheli. 2018. Deep Learning for Logic Optimization Algorithms. In *Proceedings of 2018 IEEE International Symposium on Circuits and Systems (ISCAS)*. 1–4.

[15] Aric A. Hagberg, Daniel A. Schult, and Pieter J. Swart. 2008. Exploring Network Structure, Dynamics, and Function using NetworkX. In *Proceedings of the 7th Python in Science Conference*, Gaël Varoquaux, Travis Vaught, and Jarrod Millman (Eds.). Pasadena, CA USA, 11 – 15.

[16] William L. Hamilton, Rex Ying, and Jure Leskovec. 2017. Inductive Representation Learning on Large Graphs. In *Proceedings of the 31st International Conference on Neural Information Processing Systems (NIPS) (NIPS'17)*. Red Hook, NY, USA, 1025–1035.

[17] Sepp Hochreiter and Jürgen Schmidhuber. 1997. Long Short-Term Memory. *Neural Computation* 9, 8 (1997), 1735–1780.

[18] Abdelrahman Hosny, Soheil Hashemi, Mohamed Shalan, and Sherief Reda. 2020. DRiLLS: Deep Reinforcement Learning for Logic Synthesis. In *Proceedings of the 25th Asia and South Pacific Design Automation Conference (ASP-DAC)*. 581–586.

[19] Weihua Hu, Bowen Liu, Joseph Gomes, Marinka Zitnik, Percy Liang, Vijay Pande, and Jure Leskovec. 2019. Strategies for Pre-training Graph Neural Networks. *arXiv preprint arXiv:1905.12265* (2019).

[20] Jie-Hong Roland Jiang and Srinivas Devadas. 2009. Logic Synthesis in a Nutshell. In *Electronic Design Automation*. Elsevier, 299–404.

[21] Thomas N Kipf and Max Welling. 2016. Semi-Supervised Classification with Graph Convolutional Networks. *arXiv preprint arXiv:1609.02907* (2016).

[22] Rongjian Liang, Chia-Tung Ho, Anthony Agnesina, Wen-Hao Liu, and Haoxin Ren. 2024. ReLS: Retrieval Is Efficient Knowledge Transfer For Logic Synthesis. In *Proceedings of the 6th ACM/IEEE Symposium on Machine Learning for CAD (MLCAD)*. 1–7.

[23] Gai Liu and Zhiru Zhang. 2019. PIMap: A Flexible Framework for Improving LUT-Based Technology Mapping via Parallelized Iterative Optimization. *ACM Transactions on Reconfigurable Technology and Systems* 11, 4 (2019).

[24] Junfeng Liu, Liwei Ni, Lei Chen, Xing Li, Qinghua Zhao, Xingquan Li, and Shuai Ma. 2025. A Delay-driven Iterative Technology Mapping Framework. *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems* (2025).

[25] Junfeng Liu, Liwei Ni, Xingquan Li, Min Zhou, Lei Chen, Xing Li, Qinghua Zhao, and Shuai Ma. 2023. AiMap: Learning to Improve Technology Mapping for ASICs via Delay Prediction. In *Proceedings of IEEE International Conference on Computer Design (ICCD)*. IEEE, 344–347.

[26] Yanbei Liu, Yu Zhao, Xiao Wang, Lei Geng, and Zhitao Xiao. 2023. Multi-Scale Subgraph Contrastive Learning. In *Proceedings of the 32nd International Joint Conference on Artificial Intelligence (IJCAI)*. 2215–2223.

[27] Thomas W. MacFarland and Jan M. Yates. 2016. *Spearman's Rank-Difference Coefficient of Correlation*. Springer International Publishing, Cham, 249–297.

[28] Łukasz Maziarka, Tomasz Danel, Sławomir Mucha, Krzysztof Rataj, Jacek Tabor, and Stanisław Jastrzębski. 2020. Molecule Attention Transformer. *arXiv preprint arXiv:2002.08264* (2020).

[29] Giovanni De Micheli. 1994. *Synthesis and Optimization of Digital Circuits*. McGraw-Hill Higher Education.

[30] Alan Mishchenko, Satrajit Chatterjee, and Robert Brayton. 2006. DAG-aware AIG Rewriting a Fresh Look at Combinational Logic Synthesis. In *Proceedings of the 43rd annual Design Automation Conference (DAC)*. 532–535.

[31] Liwei Ni, Rui Wang, Miao Liu, Xingyu Meng, Xiaoze Lin, Junfeng Liu, Guojie Luo, Zhufei Chu, Weikang Qian, Xiaoyan Yang, Biwei Xie, Xingquan Li, and Huawei Li. 2025. 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* (2025), 1–1.

[32] Adam Paszke, Sam Gross, Francisco Massa, Adam Lerer, James Bradbury, Gregory Chanan, Trevor Killeen, Zeming Lin, Natalia Gimelshein, Luca Antiga, Alban Desmaison, Andreas Köpf, Edward Yang, Zach DeVito, Martin Raison,

Alykhan Tejani, Sasank Chilamkurthy, Benoit Steiner, Lu Fang, Junjie Bai, and Soumith Chintala. 2019. PyTorch: An Imperative Style, High-Performance Deep Learning Library. arXiv:1912.01703 [cs.LG]

[33] Tsutomu Sasao. 1993. *Logic Synthesis and Optimization*. Vol. 2. Springer.

[34] Franco Scarselli, Marco Gori, Ah Chung Tsoi, Markus Hagenbuchner, and Gabriele Monfardini. 2009. The Graph Neural Network Model. *IEEE Transactions on Neural Networks* 20, 1 (2009), 61–80.

[35] Feng Shi, Chonghan Lee, Mohammad Khairul Bashar, Nikhil Shukla, Song-Chun Zhu, and Vijaykrishnan Narayanan. 2021. Transformer-based Machine Learning for Fast SAT Solvers and Logic Synthesis. *arXiv preprint arXiv:2107.07116* (2021).

[36] Yundong Sun, Dongjie Zhu, Yansong Wang, Yansheng Fu, and Zhaoshuo Tian. 2025. GTC: GNN-Transformer co-contrastive learning for self-supervised heterogeneous graph representation. *Neural Networks* 181 (2025), 106645.

[37] Eleonora Testa, Mathias Soeken, Luca Gaetano Amar, and Giovanni De Micheli. 2019. Logic Synthesis for Established and Emerging Computing. *Proc. IEEE* 107, 1 (2019), 165–184.

[38] Luis H.M. Torres, Bernardete Ribeiro, and Joel P. Arrais. 2024. Multi-scale cross-attention transformer via graph embeddings for few-shot molecular property prediction. *Applied Soft Computing* 153 (2024), 111268.

[39] Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N Gomez, Łukasz Kaiser, and Illia Polosukhin. 2017. Attention is All You Need. *Advances in Neural Information Processing Systems* 30 (2017).

[40] Dhananiaya Wijerathne, Zhaoying Li, Anuj Pathania, Tulika Mitra, and Lothar Thiele. 2021. HiMap: Fast and Scalable High-Quality Mapping on CGRA via Hierarchical Abstraction. In *Proceedings of 2021 Design, Automation & Test in Europe Conference & Exhibition (DATE)*. 1192–1197.

[41] Clifford Wolf. 2016. Yosys open synthesis suite. (2016). https://yosyshq.net/yosys/

[42] Nan Wu, Jiwon Lee, Yuan Xie, and Cong Hao. 2022. LOSTIN: Logic Optimization via Spatio-Temporal Information with Hybrid Graph Models. In *Proceedings of the 33rd IEEE International Conference on Application-specific Systems, Architectures and Processors (ASAP)*. 11–18.

[43] Nan Wu, Yuan Xie, and Cong Hao. 2022. AI-assisted Synthesis in Next Generation EDA: Promises, Challenges, and Prospects. In *Proceedings of the 40th IEEE International Conference on Computer Design (ICCD)*. 207–214.

[44] Keyulu Xu, Weihua Hu, Jure Leskovec, and Stefanie Jegelka. 2018. How Powerful are Graph Neural Networks? *arXiv preprint arXiv:1810.00826* (2018).

[45] Xiaoqing Xu, Nishi Shah, Andrew Evans, Saurabh Sinha, Brian Cline, and Greg Yeric. 2017. Standard cell library design and optimization methodology for ASAP7 PDK. In *Proceedings of 2017 IEEE/ACM International Conference on Computer-Aided Design (ICCAD)*. IEEE, 999–1004.

[46] Chenghao Yang, Zhongda Wang, Yinshui Xia, and Zhufei Chu. 2022. The Prediction of The Quality of Results in Logic Synthesis Using Transformer and Graph Neural Networks. *arXiv preprint arXiv:2207.11437* (2022).

[47] Chenghao Yang, Yinshui Xia, Zhufei Chu, and Xiaojing Zha. 2022. Logic Synthesis Optimization Sequence Tuning Using RL-Based LSTM and Graph Isomorphism Network. *IEEE Transactions on Circuits and Systems II: Express Briefs* 69, 8 (2022), 3600–3604.

[48] Cunxi Yu, Houping Xiao, and Giovanni De Micheli. 2018. Developing Synthesis Flows Without Human Knowledge. In *Proceedings of the 55th ACM/IEEE Design Automation Conference (DAC)*. 1–6.

[49] Jianyong Yuan, Peiyu Wang, Junjie Ye, Mingxuan Yuan, Jianye Hao, and Junchi Yan. 2023. EasySO: Exploration-enhanced Reinforcement Learning for Logic Synthesis Sequence Optimization and a Comprehensive RL Environment. In *Proceedings of the 42nd IEEE/ACM International Conference on Computer Aided Design (ICCAD)*. 1–9.

[50] Peiyan Zhang, Yuchen Yan, Chaozhuo Li, Senzhang Wang, Xing Xie, and Sunghun Kim. 2023. Can Transformer and GNN Help Each Other? *arXiv preprint arXiv:2308.14355* (2023).

[51] Keren Zhu, Mingjie Liu, Hao Chen, Zheng Zhao, and David Z. Pan. 2020. Exploring Logic Optimizations with Reinforcement Learning and Graph Convolutional Network. In *Proceedings of the 2nd ACM/IEEE Workshop on Machine Learning for CAD (MLCAD)*. 145–150.

Received 14 January 2025; revised 17 May 2025; accepted 23 July 2025