DrlGoFPGA 2.0: FPGA Global Placement Based on Multi-Agent Graph Transformers and Nonlinear Placement Co-Optimization

Kang Yang, Jianwang Zhai, Liuyu Xiang, Kang Zhao, Wei Li, Ming Lei, Zhaofeng He

***Abstract*—The speed and quality of global placement (GP) are crucial for FPGA performance. The existing nonlinear placers provide higher computational efficiency than heuristic algorithms, but their insufficient attention to the initialization of GP distributions limits rapid convergence towards high-quality global optimal solutions. To resolve this, we propose a GP framework, DrlGoFPGA 2.0, combining LUT, FF, DSP, and RAM (LFDR) instance placement generation based on multi-agent graph transformer network (MAGTN) and LFDR initial placement fine-tuning based on nonlinear placement. A MAGTN model for LFDR initial placement is designed, which utilizes distributed training to circumvent the issues of sub-optimal placement models and prolonged training times due to the non-uniform distribution of large-scale LFDR sites. A lightweight LFDR line-network relationships (LNR) graph supporting MAGTN co-decision is created by separating, renumbering, and deduplicating the LNR graph containing all instances, and inputting the LFDR LNR graph into MAGTN to obtain the connections between LFDRs. The experimental results on ISPD'2016/2017 benchmarks show that compared with the state-of-the-art (SOTA) nonlinear placer, DrlGoFPGA 2.0 can improve GP speed by 27.3% $\sim$ 82.1%, reduce half-perimeter wirelength by 0.6% $\sim$ 4.6%, and routed wirelength by 0.4% $\sim$ 2.8%.**

***Index Terms*—FPGA global placement, multi-agent, graph transformer network, nonlinear placer.**

I. Introduction

THE filed programmable gate arrays (FPGAs) demonstrate superiority over general-purpose processors in numerous scenarios due to their parallelism and customizability [1]. Placement is a critical step in FPGA physical design, significantly influencing final quality. Global placement is essential for achieving high-quality results in subsequent legalization (LG) and detailed placement (DP). However, the GP of FPGA faces challenges due to architectural heterogeneity, requiring complex constraints for site-specific packaging [2], and the non-uniform distribution of varied resource types (e.g., CLB, DSP, BRAM, I/O) [3]. These factors considerably impact the efficiency and optimality of GP algorithms.

This work is supported by the National Natural Science Foundation of China (No. 62404021, 62576046, 62301066, 62406028), the Beijing Academy of Artificial Intelligence (Z251100008125041), the Key Project of Philosophy and Social Sciences Research, Ministry of Education, China (No.24JZD040), the Beijing University of Posts and Telecommunications (2025YZ010), Beijing Natural Science Foundation (L259043), the State Key Lab of Processors, Institute of Computing Technology, CAS (No. CLQ202504), the Fundamental Research Funds for the BUPT (No. 2025AI4S20), and the Research Initiation Project for Introduced Talents of BUPT (No. 510124058, 510224062). (Corresponding authors: Jianwang Zhai, Zhaofeng He.)

Kang Yang, Jianwang Zhai, Kang Zhao, and Ming Lei are with the School of Integrated Circuit, Beijing University of Posts and Telecommunications, Beijing 100876, China (e-mail: yangkang@bupt.edu.cn; zhaijw@bupt.edu.cn; zhaokang@bupt.edu.cn; mlei@bupt.edu.cn).

Liuyu Xiang and Zhaofeng He are with the School of Artificial Intelligence, Beijing University of Posts and Telecommunications, Beijing 100876, China. (e-mail: xiangly@bupt.edu.cn; Zhaofenghe@bupt.edu.cn).

Wei Li is with the State Key Lab of Processors, Institute of Computing Technology, Chinese Academy of Sciences, Beijing 100876, China. (e-mail: liwei2017@ict.ac.cn).

A. Related Works

To achieve the high-quality and high-efficiency physical design of the FPGA, extensive and in-depth research has been conducted on the GP algorithm of the FPGA in the past few decades [4], [5], [6], [7], [8], [9], [10], [11], [12]. In traditional heuristic methods, Chen et al. [4] proposed simulated placement with clustering and duplication (SPCD), which integrates clustering, duplication, and placement to optimize FPGA physical design, simultaneously improving wirelength and timing. VTR 8 [5], an open-source FPGA framework, enhances placement quality and convergence through simulated annealing (SA), density-aware wire weighting, adaptive cell diffusion, and progressive legalization. However, with the increasing complexity and high integration of modern FPGAs, such heuristic algorithms suffer from exponentially rising computational costs and low efficiency in handling large-scale global placement problems.

Compared to traditional heuristic placement algorithms, nonlinear analysis placement methods can achieve better placement quality in a shorter running time. RippleFPGA [6] introduces a routing-driven analytical placer for large-scale heterogeneous FPGAs. RippleFPGA combines congestion sensing and estimation models to effectively optimize wirelength and routing congestion, verifying the effectiveness of analytical placement in the FPGA. GPlace [7] develops congestion-aware tools (GPlace-pack and GPlace-flat), achieving 5.3× faster placement with 22.5% shorter wirelength. AMF-Placer [8] targets macro placement with density-aware weighting, adaptive diffusion, and progressive legalization, enhancing quality and convergence.

In recent years, a novel nonlinear placement algorithm has achieved faster and more robust numerical convergence in FPGA placement tasks. The elfplace [9] is a nonlinear placement based on gradient optimization, which analogizes placement problems to electrostatic systems, extends the ePlace [13] to handle heterogeneous resources in FPGA design, and uses enhanced Lagrange formula, preprocessing techniques, and normalized subgradient methods to achieve fast and robust convergence. The elfPlace significantly improves the running speed of GP through GPU acceleration technology. DREAMPlaceFPGA [10], [11] is an open-source FPGA placement framework based on deep learning toolkits, which handles

heterogeneous resources and architecture-specific legitimacy constraints by developing new operators. DREAMPlaceFPGA is 5.4× faster and achieves comparable placement quality than elfPlace-CPU in the GP phase, and 19% faster than elfPlace-GPU. OpenPARF [12] is an open-source FPGA placement and routing framework based on PyTorch [14], which solves placement problems by introducing asymmetric multi-electrostatic field systems. It supports fine-grained configurable logic block routing models, large-scale irregular routing resource maps, and clock routing constraints, effectively alleviating routing congestion and achieving a 0.4% $\sim$ 12.7% reduction in routed wirelength and over 2× placement acceleration.

However, the nonlinear placement mentioned above did not take into account the significant impact of the placement position of IOBUF instances on GP speed and quality. DrlGoFPGA [15] first proposed a joint optimization method for IOBUF placement based on deep reinforcement learning (DRL) and LFDR placement based on nonlinear placement. Compared with the SOTA nonlinear placer, DrlGoFPGA achieves 13.2% $\sim$ 7× GP speed acceleration, 0.2% $\sim$ 2.6% reduction in half-perimeter wirelength (HPWL), and 0.2% $\sim$ 1.5% reduction in routed wirelength.

B. Motivation and Challenges

1) Impact of LFDR initial placement on GP speed and quality: While existing nonlinear placement methods have provided viable solutions for modern FPGA and shown promise in addressing complex architectural constraints and congestion-aware optimization, they exhibit a critical limitation: insufficient attention to the strategic initialization of placement distributions. This neglect fundamentally restricts their capacity to escape local optima and rapidly converge toward high-quality global solutions. Overcoming this deficiency could open new pathways for significant improvements in both solution quality and computational efficiency.

2) Large-scale and non-uniform distribution LFDR placement task challenge: The joint optimization method of IOBUF placement based on DRL and LFDR placement based on the nonlinear placement achieved good GP results in DrlGoFPGA. However, the scale of LFDR instance is hundreds of times larger than the IOBUF instance scale, and the LFDR sites exhibit non-uniform distribution in the FPGA design, making it difficult for the DrlGoFPGA method to be applied to LFDR instance placement tasks.

3) Large-scale LFDR graph learning challenge: The IOBUF LNR graph in DrlGoFPGA is a method of determining the connection between IOBUFs by identifying at least one identical in the nets connected to them, and using graph neural networks to learn the IOBUF LNR graph. The ability of DrlGoFPGA to quickly obtain good IOBUF placement and GP results validates that this IOBUF LNR graph construction method is beneficial for improving the ability of agent to explore optimal solutions in DRL. However, the scale of the nets connected to the LFDR is very large, resulting in the size of the LFDR LNR graph obtained based on the IOBUF LNR graph construction method exceeding several hundred GB. Such a large LFDR LNR graph not only makes it difficult for the agent to explore the optimal solution, but also places very high demands on hardware storage capacity.

C. Contributions

DrlGoFPGA improves GP speed and quality by optimizing the IOBUF placement. In this work, we propose DrlGoFPGA 2.0, a GP framework based on MAGTN and the nonlinear placement, aimed at further improving GP optimization performance by generating better LFDR initial distribution solutions. Our key contributions are as follows:

- We present DrlGoFPGA 2.0, a GP framework that incorporates a MAGTN for distributed generation of LFDR initial placement positions and a nonlinear placement for precise positional fine-tuning of the initial placements.
- We design a MAGTN model for LFDR initial placement, which solves the challenges of complex interconnection relationships and non-uniform constraints in large-scale design through distributed training of the LFDR placement model, while significantly reducing training time.
- We propose a method for creating lightweight LFDR LNR graph. By employing graph reconstruction and redundant edge removal, this approach significantly reduces storage requirements while effectively preserving key topological features, thereby providing feasible technical support for MAGTN co-decision.
- We conducted experiments on ISPD'2016/2017 benchmarks, and the results showed that DrlGoFPGA 2.0 improved GP speed by about $27.3\% \sim 82.1\%$, while HPWL and routed wirelength decreased by about 0.6% $\sim$ 4.6% and 0.4% $\sim$ 2.8%.

II. Preliminaries

In this section, we briefly introduce the DrlGoFPGA, the LFDR initialization method in the nonlinear placer, the graph transformer network (GTN), and the problem formulation.

A. DrlGoFPGA

DrlGoFPGA [15] is a GP framework that combines IOBUF placement based on DRL with LFDR placement based on nonlinear placer (e.g., DREAMPlaceFPGA [10] and OpenPARF[12]). For the first time, it optimizes IOBUF placement by designing a multi-action sampling policy network, a parallelizable reward function, an IOBUF LNR graph construction method based on the instance netlist, and an IOBUF placement legalization method. DrlGoFPGA is mainly developed for Xilinx UltraScale architecture [16] and can be extended to *UltraScale+ xcvu3p ffvc1517-1-i* devices [17]. In DrlGoFPGA, the scale of the IOBUF instance is relatively small, and optimizing the IOBUF placement by DRL is a single task. The agent only needs to consider the architectural constraints of the IOBUF site. The size of LFDR instances is hundreds of times larger than that of IOBUF instances, and LFDR sites exhibit a non-uniform distribution in the FPGA. Therefore, the placement task of LFDR places higher demands on the algorithm optimization speed and ability to converge to the global optimal solution. To further improve the GP

performance of DrlGoFPGA, our study found that the initial placement distribution of LFDR has a significant impact on the solution results and convergence speed of the nonlinear placement. Therefore, we have developed DrlGoFPGA 2.0, aimed at utilizing AI technology to obtain an end-to-end LFDR initial placement generation model to assist nonlinear placers, in order to improve the solving performance and convergence speed of nonlinear placement.

B. LFDR Initialization Method in Nonlinear Placer

Compared to traditional methods (e.g., VTR [5]), nonlinear placers can achieve better placement results in a shorter time. RippleFPGA [6] and GPlace [7] have achieved better or equivalent wirelength in most circuits and significantly reduced congestion, demonstrating the effectiveness of nonlinear placement in FPGA. Subsequently, researchers developed a new generation of nonlinear placer based on gradient optimization called elfplace [9] by analogizing placement problems to electrostatic systems and achieved significant improvements in placement speed and quality. DREAMPlaceFPGA [10] and OpenPARF [12] have emerged successively based on elfplace, both of which are suitable for elfplace on PyTorch deep learning toolkit [14]. Compared with elfplace-CPU and elfplace-GPU, DREAMPlaceFPGA and OpenPARF have achieved more significant improvements in placement speed, as well as better or equivalent placement quality and routed wirelength. The elfPlace, DREAMPlaceFPGA, and OpenPARF all start from a random initial placement. In the random initial placement, all LFDR instances are first placed at the centroid of fixed pins, and an extra Gaussian noise perturbation is injected with a standard deviation equal to 0.1% of the width and height of the placement region. Existing nonlinear placers provide feasible solutions for FPGA placement, yet their neglect of initialization strategies hinders the achievement of rapid, high-quality global optimal convergence.

C. Graph Transformer Network

In recent years, multiple studies have improved the performance of chip placement based on graph neural networks (GNNs). The GoodFloorplan [18] combines GNN with reinforcement learning, achieving a routing success rate of 98.8% and reducing wirelength by 3.8% $\sim$ 15.2%. Yiting et al. [19] developed the mixed-size placement tool Flora using a graph attention network, which reduced wirelength by 2% while reducing running time by 18%. Mohammad et al. [20] used DRL and hypergraph embedding to achieve generalized ASIC floorplanning automation, significantly improving the early design efficiency of ultra large-scale integrated circuits. Dho et al. [21] optimized the detailed placement using GNN, resulting in an average reduction of 13.73% in maximum displacement and 2.44% in wirelength. However, GNN often struggles to capture remote dependencies in graph structures due to its localized message passing mechanism, resulting in performance degradation on large-scale graphs with thousands or more nodes. GTN [22] addresses these limitations by leveraging self-attention to model interactions between any node pairs-irrespective of graph distance-effectively capturing both local and global circuit structures. This enhances representational capacity for large netlists while remaining efficient. Their parallel processing and adaptive attention also make GTN well-suited for large-scale LFDR placement.

D. Problem Formulation

DrlGoFPGA 2.0 first obtains the initial distribution by generating LFDR placement positions based on MAGTN, and then quickly fine-tunes the LFDR initial placement based on a nonlinear placer to improve GP speed and results. The optimization objective of DrlGoFPGA 2.0 is consistent with that of DrlGoFPGA [15], both of which aim to minimize the total HPWL $W(x,y)$ of the GP, i.e.,

$$W(x,y)=\sum_{e\in E}\left(\max_{i,j\in E}|x_i-x_j|+\max_{i,j\in E}|y_i-y_j|\right), \quad (1)$$

where the HPWL of a net e is half the perimeter of the smallest bounding box enclosing all the pins (i,j) of the connected instances.

DrlGoFPGA 2.0 uses the SOTA nonlinear placement method based on gradient optimization (e.g., DREAMPlaceFPGA) for LFDR initial placement fine-tuning to ensure that all instances quickly meet density and other constraints. DREAMPlaceFPGA is an elfPlace [9] for PyTorch deep learning toolkit [14], which analogizes placement problems to electrostatic systems, and its optimization objective function $f(x,y)$ is defined as:

$$\min_{x,y} f(x,y)=\tilde{W}(x,y)+\sum_{s\in S}\lambda_s\left[\Phi_s(x,y)+\frac{c_s}{2}\Phi_s(x,y)^2\right], \quad (2)$$

$$\tilde{W}(x,y)=\sum_{e\in E}\left(\tilde{W}_{e_x}+\tilde{W}_{e_y}\right), \quad (3)$$

$$\tilde{W}_{e_x}=\frac{\sum_{i\in e}x_i\exp\left(\frac{x_i}{\gamma}\right)}{\sum_{i\in e}\exp\left(\frac{x_i}{\gamma}\right)}-\frac{\sum_{i\in e}x_i\exp\left(\frac{-xi}{\gamma}\right)}{\sum_{i\in e}\exp\left(\frac{-x_i}{\gamma}\right)}, \quad (4)$$

where $\tilde{W}(x,y)$ is a weighted-average (WA) model [23] that approximate the HPWL. $\tilde{W}_{e_x}$ is the x-directed WA wirelength model. $\Phi_s(x,y)$ denotes electrical potential energy that approximate the density. λ_s is the density multiplier. c_s is used to weight the quadratic penalty term $\Phi_s(x,y)^2$. γ controls the accuracy and smoothness of the HPWL approximation.

III. Implementation of DrlGoFPGA 2.0

In this section, we provide a detailed introduction to the overall flow, the MAGTN model for LFDR initial placement, LFDR LNR graph supporting MAGTN co-decision, and the MAGTN and nonlinear placement co-optimization.

A. Overall Flow

The overall workflow of DrlGoFPGA 2.0 is shown in Fig. 1, consisting of an IOBUF placement agent and LFDR placement agents. In Stage 1, the IOBUF placement agent adopts the DRL strategy from DrlGoFPGA [15] to generate IOBUF locations. Meanwhile, the LFDR instance numbers and LNR

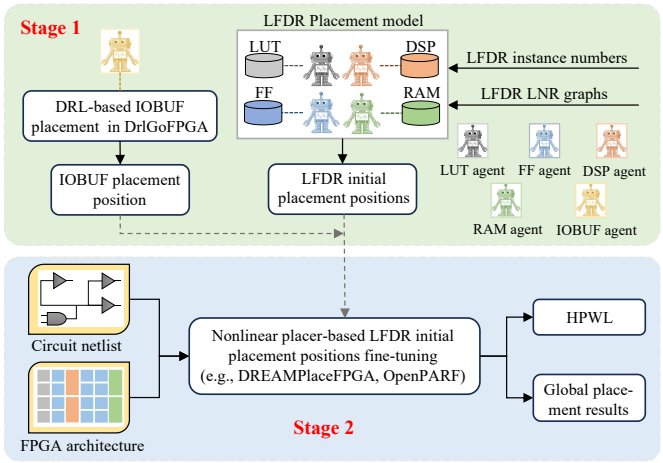

Fig. 1. FPGA global placement flow of DrlGoFPGA 2.0.

graph are fed into the LFDR agent, which produces initial LFDR placements via the LFDR placement model trained with MAGTN (Section III-B). In Stage 2, the placement information (circuit netlist and FPGA architecture), along with the IOBUF and LFDR initial placement positions obtained in Stage 1, are used as inputs to the nonlinear placer. The nonlinear placer fine-tunes the initial LFDR locations to output the final global placement result and HPWL. A multi-agent paradigm is adopted to enable concurrent inference of the IOBUF and LFDR placement models, reducing model inference latency.

B. MAGTN Model for LFDR Initial Placement

FPGAs comprise massive LFDR instances on the scale of tens of thousands to millions, with highly complex and tangled interconnections. Additionally, the LUT, FF, DSP, and RAM sites in the FPGA exhibit a non-uniform distribution in the placement [3]. If a GTN model is used to train initial placement tasks for LFDR instances, not only will a good placement model not be obtained, but it will also require an expensive training time. To address this issue, we propose an LFDR initial placement based on MAGTN model, as shown in Fig. 2, which utilizes four agents to construct independent GTNs for LUT, FF, DSP, and RAM instances separately.

In Fig. 2, GTN consists of one Embedding layer, two TransformerConv [24] layers, and one ReLU [25] activation layer. LFDR agents are responsible for handling specific types of instances. The network input is a node ID sequence $[0, 1, 2, \ldots, n]$ (n is the total number of instances for each type and $n \in \{n_{LUT}, n_{FF}, n_{DSP}, n_{RAM}\}$) composed of LFDR instance numbers, which is transformed into a 128-dimensional feature vector through the Embedding layer. Subsequently, the TransformerConv layer is used to extract features from the LFDR LNR graph, and the nonlinear expression ability is enhanced through the ReLU activation function. The final output layer generates two-dimensional placement coordinate prediction values. The agent training adopts the supervised learning paradigm, using the LFDR placement coordinates optimized by DrlGoFPGA [15] as label data, and optimizing network parameters by minimizing the mean square error (MSE) loss function:

$$\begin{aligned} \mathcal{L}_{\text{regularized}} &= \mathcal{L}_{MSE} + \frac{\lambda}{2}, \\ \mathcal{L}_{MSE} &= \frac{1}{N}\sum_{i=1}^{N}\left[(x_i', y_i') - (x_i, y_i)\right]^2, \\ (x_i', y_i') &\in \{(\boldsymbol{X}_l^p, \boldsymbol{Y}_l^p), (\boldsymbol{X}_f^p, \boldsymbol{Y}_f^p), (\boldsymbol{X}_d^p, \boldsymbol{Y}_d^p), (\boldsymbol{X}_r^p, \boldsymbol{Y}_r^p)\}, \\ (x_i, y_i) &\in \{(\boldsymbol{X}_l, \boldsymbol{Y}_l), (\boldsymbol{X}_f, \boldsymbol{Y}_f), (\boldsymbol{X}_d, \boldsymbol{Y}_d), (\boldsymbol{X}_r, \boldsymbol{Y}_r)\}, \end{aligned} \tag{5}$$

where $\mathcal{L}_{\text{regularized}}$ is a loss function consisting of MSE $\mathcal{L}_{MSE}$ and a penalty term proportional to the sum of squared weights. λ is the weight decay coefficient. w_i is the weight parameter of the model. $\sum_i w_i^2$ is the sum of the squares of all weights. (x_i', y_i') is the predicted node feature of LFDR obtained by

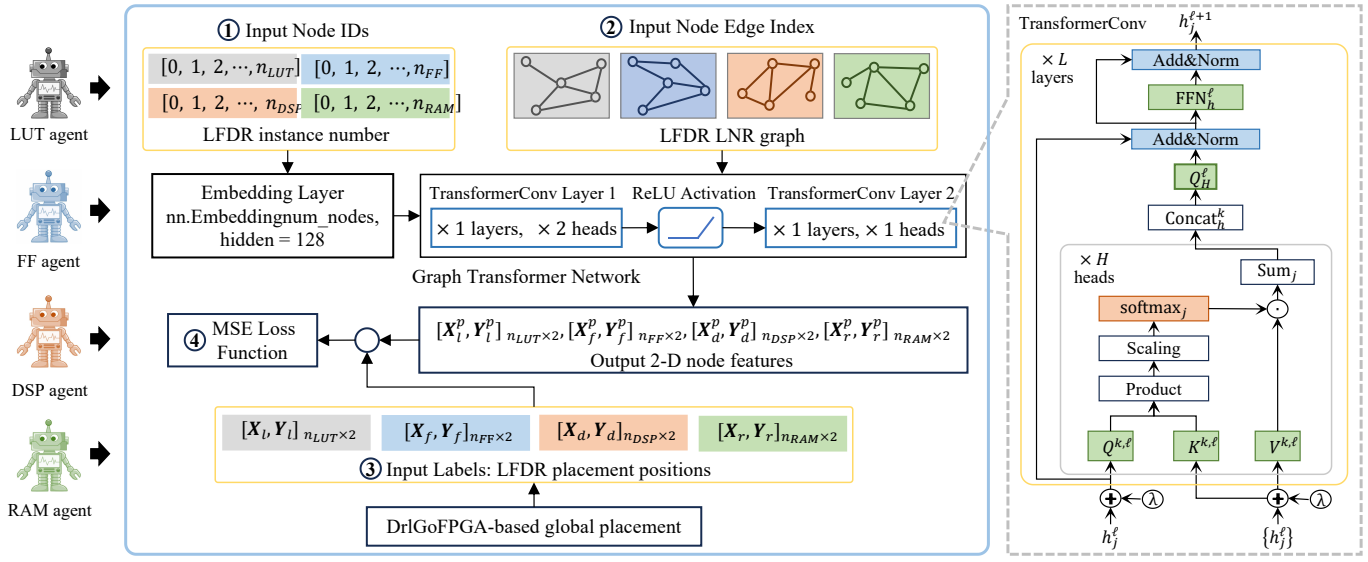

Fig. 2. LFDR initial placement based on MAGTN model.

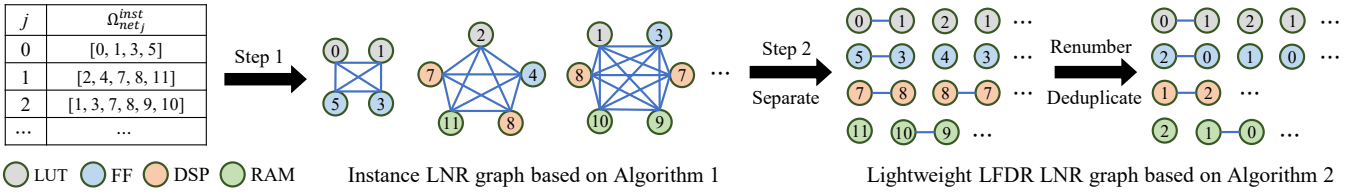

Fig. 3. Lightweight LFDR LNR graph construction flow

GTN. (x_i, y_i) is the GP coordinate of LFDR obtained after performing GP optimization based on DrlGoFPGA.

Due to the LFDR placement model using the GP coordinates generated by DrlGoFPGA as labels during training, which already satisfy various constraints. Therefore, the LFDR placement model can implicitly learn placement distributions that satisfy the same constraints from labels, and capture the topological connection relationships between instances through GTN to generate LFDR initial placement distributions that are more in line with circuit connection characteristics. Each agent learns to approximate the global optimal solution through label-guided training, thereby aligning their individual optimization objectives intrinsically and minimizing conflicts in the guidance direction from the source. We use the AdamW optimizer [26] for MAGTN parameter updates.

C. LFDR LNR Graph Supporting MAGTN Co-Decision

In FPGA design, LFDR instances have large-scale and complex connections. If the created LFDR LNR graph is very large and input into MAGTN, it will prevent MAGTN from implementing co-decision due to hardware resource storage capacity limitations. Therefore, to support MAGTN co-decision, we propose a lightweight LFDR line-network relationship (LNR) graph construction method. The "line" refers to the connection edges between each instance, while "network" refers to the vast network of connections formed by the connection edges between all instances. This method is based on the PyTorch Geometric framework [27], using the source target node representation of $[S, T]$, and completing graph data construction in two steps. The implementation of the lightweight LFDR LNR graph is shown in Fig. 3. First, create the LNR graph that includes all instances according to Algorithm 1. Then, according to Algorithm 2, separate the LNR graph containing all instances into LFDR LNR graphs, and renumber the instance numbers from 0 in ascending order, and the edges with duplicate connections are removed. From Fig. 3, the lightweight LFDR LNR graph is a topological structure composed of all instances in each net connected from left to right, rather than defining the connection relationship between two instances through pin connections. Therefore, if some nets have multiple identical instances, there may be duplicate connections, as shown in "7" and "8" in Fig. 3.

In Algorithm 1, if the number of elements of $\mathbf{\Omega}^{inst}_{net_j}$ ($\mathbf{\Omega}^{inst}_{net_j}$ is the corresponding instance number set in the j-th net) is 1, it indicates that the node is a self-looping node. At this point, simply write the node numbers separately in S_{inst} and T_{inst} (Lines 4 $\sim$ 6). If the number of $\mathbf{\Omega}^{inst}_{net_j}$ is greater than 1, the nodes are connected sequentially from beginning to end, and

Algorithm 1: Instance LNR graph construction

Input: The corresponding instance number set $\mathbf{\Omega}^{inst}_{net_j}$ in the j-th net
. **Output:** The instance LNR graph $[S_{inst}, T_{inst}]$.
$S_{inst} \leftarrow [\,]$, $T_{inst} \leftarrow [\,]$;
for $j = 0$ **to** m. **do**
  if $\left|\mathbf{\Omega}^{inst}_{net_j}\right| == 1$ **then** // $|\cdot|$ represents the total number of calculation elements
    $S_{inst} \leftarrow S_{inst} \cup \mathbf{\Omega}^{inst}_{net_j}[0]$;
    $T_{inst} \leftarrow T_{inst} \cup \mathbf{\Omega}^{inst}_{net_j}[0]$;
  else
    for $u = 0$ **to** $\left|\mathbf{\Omega}^{inst}_{net_j}\right| - 1$ **do**
      $S_{inst} \leftarrow S_{inst} \cup \mathbf{\Omega}^{inst}_{net_j}[u]$;
      $T_{inst} \leftarrow T_{inst} \cup \mathbf{\Omega}^{inst}_{net_j}[u+1]$;
      $S_{inst} \leftarrow S_{inst} \cup \mathbf{\Omega}^{inst}_{net_j}[u+1]$;
      $T_{inst} \leftarrow T_{inst} \cup \mathbf{\Omega}^{inst}_{net_j}[u]$;

Algorithm 2: Lightweight LFDR LNR graph construction

Input: The instance LNR graph $[S_{inst}, T_{inst}]$. LUT, FF, DSP, and RAM instance number values $\mathbf{ID}_{LFDR}$ in FPGA design, $LFDR \in \{LUT, FF, DSP, RAM\}$.
Output: The LFDR LNR graph $[S_{LFDR}, T_{LFDR}]$.
// Note: the following code is executed separately for each instance type;
$S_{LFDR} \leftarrow [x | x \in S_{inst} \wedge x \in \mathbf{ID}_{LFDR}]$;
$T_{LFDR} \leftarrow [x | x \in T_{inst} \wedge x \in \mathbf{ID}_{LFDR}]$;
$I \leftarrow \text{sorted}(\mathbf{ID}_{LFDR})$; // sorted$(\cdot)$ means to sort the instance numbers in ascending order ;
Define number mapping set $f^{ID}_{map} : I = [v_0, v_1, \ldots, v_{|I|-1}] \rightarrow [0, 1, \ldots, |I| - 1]$;
for $i = 0$ **to** $|S_{LFDR}| - 1$ **do**
  $S_{LFDR}[i] \leftarrow f^{ID}_{map}(S_{LFDR}[i])$; // Renumber instances from 0 in ascending order ;
  $T_{LFDR}[i] \leftarrow f^{ID}_{map}(T_{LFDR}[i])$;
$E = \{(s_i, t_i) | s_i \in S_{LFDR}, t_i \in T_{LFDR}, i = 1, \ldots, |S_{LFDR}| - 1\}$;
$E' = \{(s,t) | (s,t) \in E, \text{and } (s,t) \text{ is unique}\}$; // Remove duplicate edge connections;
$S_{LFDR} = \{s | (s,t) \in E'\}$, $T_{LFDR} = \{t | (s,t) \in E'\}$;

the numbers between nodes are written in S_{inst} and T_{inst} in a bidirectional connection (Lines 8 $\sim$ 12). In Algorithm 2, it is necessary to create the LFDR LNR graphs in sequence. We will use an example of LUT to illustrate this. Firstly, extract all the numbers belonging to the LUT instance number values $\mathbf{ID}_{LUT}$ from the S_{inst} and T_{inst}, and form a new S_{LUT} and

TABLE I
COMPARISON OF LFDR LNR GRAPH SIZE BEFORE AND AFTER LIGHTWEIGHTING OF ISPD'2016/2017 BENCHMARKS

Design	Non-lightweight LFDR LNR graph size					Lightweight LFDR LNR graph size				
	LUT (MB)	FF (GB)	DSP (KB)	RAM (KB)	Tot (GB)	LUT (MB)	FF (MB)	DSP (KB)	RAM (KB)	Tot (GB)
FPGA01	7.6	20.2	-	-	16700.8	10.7	2.4	-	-	1.0
FPGA02	25.2	29.8	77.20	77.20		7.2	3.0	1.8	3.0	
FPGA03	80.4	207.2	43.00	7168.0		20.6	10.6	21.6	26.0	
FPGA04	77.2	216.4	103.20	7168.0		21.6	10.8	39.8	26.4	
FPGA05	103.4	223.7	201.00	7168.0		22.6	11.0	50.8	26.0	
FPGA06	65.6	953.6	294.00	20070.4		27.2	22.0	80.0	41.0	
FPGA07	68.6	969.9	211.40	20070.4		28.8	22.4	61.6	41.6	
FPGA08	315.0	350.1	25.60	7168.0		46.4	11.8	16.8	23.0	
FPGA09	138.4	1029.5	203.80	20070.4		45.2	22.6	64.8	41.0	
FPGA10	33.2	2969.6	455.00	20070.4		21.6	33.2	99.4	32.4	
FPGA11	133.0	1015.1	85.80	20070.4		43.2	19.4	30.4	32.8	
FPGA12	83.0	2867.2	347.40	7168.0		36.8	29.6	76.8	17.4	
CLK-FPGA01	26.2	202.4	12.0	54.2		13.8	13.2	2.4	3.2	
CLK-FPGA02	55.2	156.8	23.6	144.2		16.6	10.8	3.2	4.6	
CLK-FPGA03	86.0	214.3	190.4	924.2		31.0	19.4	20.0	19.8	
CLK-FPGA04	65.4	212.1	86.6	537.8		23.0	14.8	8.6	11.0	
CLK-FPGA05	87.6	308.2	64.2	1231.8		29.6	18.6	3.6	18.6	
CLK-FPGA06	89.4	231.7	253.0	1020.8		31.8	20.6	23.0	20.0	
CLK-FPGA07	60.8	183.1	43.2	238.2		18.2	11.8	5.0	6.4	
CLK-FPGA08	40.0	141.6	12.2	59.2		15.2	10.4	2.4	3.2	
CLK-FPGA09	33.4	230.9	23.0	129.0		15.4	14.6	2.8	4.6	
CLK-FPGA10	51.6	408.2	174.0	636.0		22.2	20.8	14.4	12.4	
CLK-FPGA11	47.0	298.8	85.2	443.0		20.4	19.6	10.2	10.4	
CLK-FPGA12	43.8	332.9	128.6	363.4		18.6	17.8	7.8	9.0	
CLK-FPGA13	71.0	282.0	182.4	734.0		25.4	15.8	10.6	13.2	

T_{LUT} (Lines 1 $\sim$ 2). Due to the fact that in FPGA design, instance numbers are not necessarily sorted sequentially for each instance type, starting from 0. Therefore, it is necessary to sort the instance numbers in ascending order (Line 3), and then establish a number mapping set f_{map}^{ID} starting from 0 (Line 4). According to f_{map}^{ID}, renumber S_{LUT} and T_{LUT} (Lines 6 $\sim$ 7). Finally, remove duplicate edge connections in $[S_{LUT}, T_{LUT}]$ to obtain simplified S_{LUT} and T_{LUT} (Lines 9 $\sim$ 10).

The comparison of LFDR LNR graph size before and after lightweighting of ISPD'2016/2017 benchmarks is shown in Table I. The dataset size range of our proposed lightweight LFDR LNR graph construction method is 1.8 KB to 46.4 MB, and its total size is about 16,701$\times$ smaller than before lightweighting, fully supporting MAGTN co-decision. This also provides a feasible technical path for MAGTN to expand to larger-scale FPGA and other chip placement design optimization.

D. MAGTN and Nonlinear Placement Co-Optimization

In MAGTN, obtaining the optimal LFDR placement strategy directly through end-to-end training poses significant challenges due to the large-scale and complex connections of LFDR instances. Therefore, DrlGoFPGA 2.0 adopts a co-optimization strategy of MAGTN and nonlinear placement: the MAGTN is responsible for generating high-quality LFDR initial placements, and the nonlinear placement is responsible for fine-tuning the LFDR initial placements. This division of labor mechanism fully utilizes their respective advantages: MAs capture complex topological relationships through GTN and generate initial placements that conform to circuit connection characteristics; Nonlinear placement performs local optimization based on physical constraints. We chose DREAMPlaceFPGA [10] and OpenPARF [12] as the engines for LFDR placement fine-tuning, which are superior to traditional heuristic algorithms in terms of computational efficiency and solution quality.

IV. EXPERIMENTAL RESULTS

A. Benchmarks and Experimental Settings

We conduct experiments using the ISPD'2016 [28] and ISPD'2017 [29] benchmarks. Table II and Table III show the composition of the benchmarks. All experiments are run on a Linux server that consists of an Intel (R) Core (TM) i9-10920X CPU @ 3.50 GHz (12 cores) and 2 NVIDIA Corporation GP102 [GeForce RTX 3090] GPUs. PyTorch [14] is used for all experiments. The LFDR placement model training based on MAGTN uses the AdamW optimizer [26] for parameter updates. Table V compares the GP and routing results of DrlGoFPGA 2.0 with DREAMPlaceFPGA [10] and DrlGoFPGA [15]. Table VI and Table VII compare the GP and routing results of DrlGoFPGA 2.0 with OpenPARF [12] and DrlGoFPGA [15]. As DREAMPlaceFPGA does not support clock routing constraints on ISPD'2017 benchmarks, we do not include it in the comparison. In all comparison metrics, "RWL" is the wirelength after routing the placed designs using the router in OpenPARF. "HPWL" is the half-perimeter wirelength after GP. "GPRT" is GP runtime. "ORT" is the overall runtime of GP, LG, and DP. In DrlGoFPGA 2.0, GPRT is composed of IOBUF placement model inference time, LFDR placement model inference time, and LFDR placement position fine-tuning time. In DrlGoFPGA, GPRT is composed of the IOBUF placement model inference time and LFDR placement time based on the nonlinear placer.

B. Quantitative Analysis of Lightweight LFDR LNR Graph

To quantitatively analyze the lightweight LFDR LNR graph, we calculated the edge retention rate (ERR) and edge loss

TABLE II
THE COMPOSITION OF ISPD'2016 BENCHMARKS [28]

Design	#LUT/#FF/#DSP/#RAM/#IOBUF	#Net
FPGA01	50K/55K/0/0/151	105K
FPGA02	100K/66K/100/100/151	168K
FPGA03	250K/170K/500/600/401	429K
FPGA04	250K/172K/500/600/401	430K
FPGA05	250K/174K/500/600/401	433K
FPGA06	350K/352K/600/1000/601	713K
FPGA07	350K/355K/600/1000/601	716K
FPGA08	500K/216K/500/600/401	725K
FPGA09	500K/366K/600/1000/601	877K
FPGA10	350K/600K/600/1000/601	961K
FPGA11	480K/363K/400/1000/601	851K
FPGA12	500K/602K/500/600/401	1111K

TABLE III
THE COMPOSITION OF ISPD'2017 BENCHMARKS [29]

Design	#LUT/#FF/#DSP/#RAM/#IOBUF	#Net	#Clock
CLK-FPGA01	211K/324K/75/164/331	536K	32
CLK-FPGA02	230K/280K/112/236/335	512K	35
CLK-FPGA03	410K/481K/395/850/444	899K	57
CLK-FPGA04	309K/372K/224/467/434	685K	44
CLK-FPGA05	393K/469K/150/798/444	866K	56
CLK-FPGA06	425K/511K/420/872/444	943K	58
CLK-FPGA07	254K/309K/149/313/338	565K	38
CLK-FPGA08	212K/257K/75/161/332	471K	32
CLK-FPGA09	231K/358K/112/236/335	592K	35
CLK-FPGA10	327K/506K/255/542/434	838K	47
CLK-FPGA11	300K/468K/224/454/427	773K	44
CLK-FPGA12	277K/430K/187/389/339	710K	41
CLK-FPGA13	339K/405K/262/570/437	750K	47

rate (ELR) after lightweighting, as shown in Eq. (6) and (7), respectively.

$$\mathrm{ERR} = \frac{|E_{lw}|}{|E_{lw}^{un}|} \times 100\%, \tag{6}$$

$$\mathrm{ELR} = \frac{|E_{lw}^{un} \setminus E_{lw}|}{|E_{lw}^{un}|} \times 100\%, \tag{7}$$

where E_{lw} and E_{lw}^{un} respectively are the edge sets in the LFDR LNR graph after and before lightweighting, and $E_{lw} \in E_{lw}^{un}$. $|\cdot|$ is the number of elements in the edge set, and $E_{lw}^{un} \setminus E_{lw}$ is the set of edges in E_{lw}^{un} but not in E_{lw}.

Additionally, to measure the structural preservation and degree distribution differences of lightweight LFDR LNR graph compared to non-lightweight graph, we use Pearson correlation coefficient [34] and KL divergence [35] to calculate the degree distribution correlation (DCC) and degree distribution KL divergence (DKL), as shown in Eq. (8) and (9), respectively.

$$\mathrm{DCC} = \frac{\sum_{i=1}^{n}(d_i - \bar{d})(d_i' - \bar{d}')}{\sqrt{\sum_{i=1}^{n}(d_i - \bar{d})^2 \sum_{i=1}^{n}(d_i' - \bar{d}')^2}}, \tag{8}$$

$$\mathrm{DKL}(P \parallel Q) = \sum_{k=0}^{k_{\max}} P(k) \log_2 \frac{P(k)}{Q(k)}, \tag{9}$$

$$P(k) = \frac{|\{v_i \in V_{lw}^{un} : d_i = k\}|}{n}, \tag{10}$$

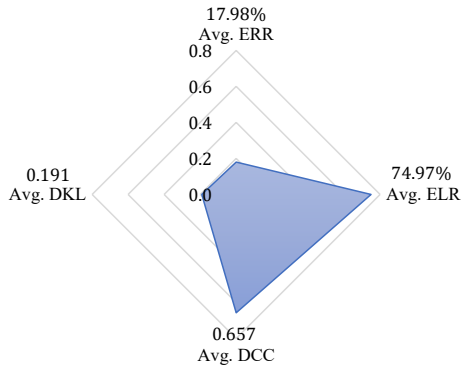

Fig. 4. The quantitative analysis of lightweight LFDR LNR graph.

$$Q(k) = \frac{|\{v_i' \in V_{lw} : d_i' = k\}|}{n}, \tag{11}$$

where d_i and d_i' are the degrees of nodes in the LFDR LNR graph before and after lightweighting. $\bar{d} = \frac{1}{n}\sum_{i=1}^{n} d_i$, $\bar{d}' = \frac{1}{n}\sum_{i=1}^{n} d_i'$. $P(k)$ and $Q(k)$ are the degree distribution probabilities of the LFDR LNR graph before and after lightweighting. $k_{\max} = \max(\max_i d_i, \max_i d_i')$, and $0\log_2(0/q) = 0$, $p\log_2(p/0) = \infty$. V_{lw}^{un} and V_{lw} are the node sets of the LFDR LNR graph before and after lightweighting.

For ISPD'2016/2017 benchmarks, Fig. 4 presents the average (Avg.) ERR, Avg. ELR, Avg. DCC, and Avg. DKL of all LFDR LNR graphs. Specifically, Avg. ERR is 17.98%, corresponding to Avg. ELR reaching 74.97%, indicating that over 70% of the original edges have been pruned and the graph scale has been significantly compressed. Avg. DCC is 0.657 (>0.6 indicates good structural preservation), reflecting a strong positive correlation between the degree distribution of nodes, indicating that the topological importance of key high connectivity nodes is preserved during the compression process. Avg. DKL is 0.190 (<0.2 indicates that the degree distribution difference is within an acceptable range), further indicating that lightweighting did not significantly distort the overall degree distribution morphology.

Overall, this lightweight strategy achieves efficient graph compression while still maintaining the core topological features of the original circuit, providing a compact and representative structural input for subsequent placement modeling based on the graph.

C. LFDR Placement Model Pre-Training and Fine-Tuning

FPGA02 on ISPD'2016 benchmarks is a relatively small design that includes LUT, FF, DSP, and RAM instances, compared to other designs. Therefore, we chose FPGA02 for pre-training the LFDR placement model. The pre-training epochs are set to 300 for LUT/FF models and 400 for DSP/RAM models. The learning rate of the AdamW optimizer is set to 10^{-3}, and the weight decay coefficient is set to 10^{-4}. Due to the dimensionality of the Embedding layer in GTN being related to the number of nodes, placement tasks with different LFDR scales require retraining the model to maintain performance. We fine-tuned the pre-trained model to obtain LFDR placement models for other designs. The initial weight

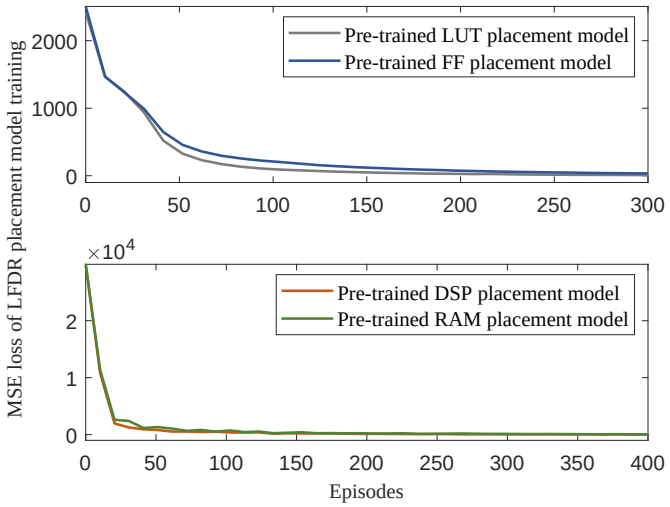

Fig. 5. The MES loss iteration of LFDR model training.

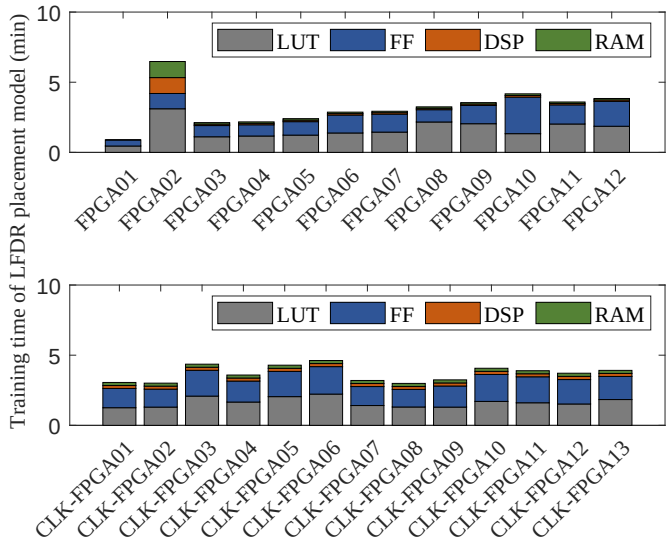

Fig. 6. The training time distribution of LFDR placement model.

parameter design for the fine-tuned model is as follows: 1) In the Embedding layer, if the number of LFDR instances in the target design is less than FPGA02, the front part of the pre-trained Embedding layer weight is truncated as initialization; If there are more, the weights of the newly added nodes are sampled from a Gaussian distribution centered on the mean weight of the pre-trained Embedding layer and with a standard deviation of 0.01. 2) The pre-training parameters of the remaining layers in GTN are directly transferred. The fine-tuning epochs are set to 150 for LUT/FF models and 200 for DSP/RAM models. The learning rate of the AdamW optimizer is set to 10^{-4}. Additionally, when using DrlGoFPGA [15] to generate LFDR placement labels, the nonlinear placer selects DREAMPlaceFPGA to achieve LFDR fine-tuning. However, due to the fact that DREAMPlaceFPGA does not support clock routing constraints, we need to replace the nonlinear placer in DrlGoFPGA with OpenPARF when fine-tuning the LFDR model on ISPD'2017 benchmarks.

TABLE IV
LFDR PLACEMENT MODEL SIZE OF ISPD'2016/2017 BENCHMARKS

Design	LFDR placement model size			
	LUT (MB)	FF (MB)	DSP (MB)	RAM (MB)
FPGA01	26.1	28.8	-	-
FPGA02	51.7	34.4	0.6	0.6
FPGA03	128.5	87.6	0.8	0.8
FPGA04	128.5	88.4	0.8	0.8
FPGA05	128.5	89.8	0.8	0.8
FPGA06	179.7	180.9	0.8	1.1
FPGA07	179.7	182.4	0.8	1.1
FPGA08	256.5	111.3	0.8	0.8
FPGA09	256.5	187.8	0.8	1.1
FPGA10	179.7	307.8	0.8	1.1
FPGA11	246.3	186.5	0.7	1.1
FPGA12	256.5	309.0	0.8	0.8
CLK-FPGA01	108.5	166.3	0.6	0.6
CLK-FPGA02	118.1	143.8	0.6	0.7
CLK-FPGA03	210.5	247.0	0.7	1.0
CLK-FPGA04	158.8	191.0	0.6	0.8
CLK-FPGA05	202.0	240.7	0.6	0.9
CLK-FPGA06	217.9	262.2	0.7	1.0
CLK-FPGA07	130.3	158.6	0.6	0.7
CLK-FPGA08	109.2	132.0	0.6	0.6
CLK-FPGA09	118.8	184.1	0.6	0.7
CLK-FPGA10	167.7	259.8	0.7	0.8
CLK-FPGA11	154.3	240.2	0.6	0.8
CLK-FPGA12	142.2	220.7	0.6	0.7
CLK-FPGA13	174.3	208.1	0.7	0.8

In Fig. 5, the pre-trained LFDR placement model can quickly converge to a stable MSE loss value during training. In Fig. 6, except for the pre-trained LFDR placement model on FPGA02, which takes about 7 minutes to train, the fine-tuned LFDR placement models on other designs are trained in about 1-5 minutes. The size of the LFDR placement model ranges from 0.6 MB to 309.0 MB, as shown in Table IV.

D. Evaluation on ISPD'2016 benchmarks

Table V and Table VI present the comparison results of DREAMPlaceFPGA, OpenPARF, DrlGoFPGA, and DrlGoFPGA 2.0 on ISPD'2016 benchmarks, respectively. DREAMPlaceFPGA uses an integrated elfplace-CPU [9] for legalization and detailed placement, and uses the router of OpenPARF for routing. OpenPARF has integrated corresponding legalization, detailed placement, and routing tools. The results of Table V and Table VI are summarized as follows:

- In Table V, compared to DREAMPlaceFPGA and DrlGoFPGA, DrlGoFPGA 2.0 achieved an 82.1% and 43.8% decrease in GPRT, a 5.5% and 4.6% decrease in ORT, a 2.0% and 1.1% decrease in HPWL, and a 1.9% and 0.8% decrease in RWL.
- In Table VI, compared to OpenPARF and DrlGoFPGA, DrlGoFPGA 2.0 achieved a 46.1% and 46.2% decrease in GPRT, a 13.7% and 12.5% decrease in ORT, a 1.8% and 0.6% decrease in HPWL, and a 1.8% and 0.4% decrease in RWL.

Therefore, the proposed DrlGoFPGA 2.0 not only improves the GP and ORT speed, but also ensures obtaining a better placement result and routed wirelength. Additionally, after using DrlGoFPGA with DREAMPlaceFPGA as the nonlinear placer to generate placement position labels for training and

TABLE V
COMPARISON OF DREAMPLACEFPGA AND DRLGOFPGA 2.0 ON ISPD'2016 BENCHMARKS (HPWL AND RWL IN 10^3, GPRT AND ORT IN SECONDS)

Design	DREAMPlaceFPGA [10]				DrlGoFPGA [15]				DrlGoFPGA 2.0			
	RWL	HPWL	GPRT	ORT	RWL	HPWL	GPRT	ORT	RWL	HPWL	GPRT (IOBUF+LFDR+NP)	ORT
FPGA01	339.4	190.9	8.7	24.4	335.5	188.7	7.2	23.8	335.2	189.2	7.0 (0.5+0.4+6.1)	24.5
FPGA02	698.8	490.4	22.8	59.6	698.0	491.5	19.7	59.4	685.6	477.6	12.8 (0.5+0.4+11.9)	47.8
FPGA03	3101.2	2053.4	21.6	197.8	3024.4	1997.0	17.2	196.3	2935.9	1940.0	11.9 (0.6+0.5+10.8)	198.3
FPGA04	5999.8	4023.7	23.1	194.9	5974.6	3976.3	17.6	192.4	5931.9	3983.3	11.4 (0.6+0.4+10.4)	182.1
FPGA05	12430.9	8122.8	26.6	205.4	12058.1	7920.1	19.9	207.4	12196.3	7976.2	11.9 (0.6+0.4+10.9)	194.9
FPGA06	6105.6	3403.3	24.1	450.8	5940.5	3349.4	19.0	447.6	5858.3	3255.9	12.5 (0.7+0.4+11.4)	425.9
FPGA07	10344.3	6214.3	24.1	457.8	10286.9	6219.7	19.1	458.7	10246.7	6208.7	11.9 (0.6+0.5+10.7)	437.1
FPGA08	9593.3	6735.4	25.0	498.5	9494.3	6606.1	19.6	502.8	9455.1	6593.9	14.4 (0.6+0.5+13.3)	482.8
FPGA09	12882.8	8132.6	26.7	756.8	12870.9	8182.6	21.3	762.0	12839.2	8162.9	14.3 (0.6+0.5+13.2)	754.0
FPGA10	5945.6	3326.8	25.6	697.9	5923.3	3309.3	19.1	664.7	5844.4	3244.2	12.2 (0.6+0.4+11.2)	645.8
FPGA11	13262.2	8921.3	25.5	697.6	13321.4	9029.3	20.6	678.8	13152.8	8986.4	16.9 (0.7+0.5+15.6)	666.9
FPGA12	7408.4	4384.6	29.2	1110.1	7292.2	4350.3	23.7	1114.4	7287.0	4238.9	18.6 (0.6+0.5+17.4)	1082.5
Ratio	1.019	1.020	1.821	1.055	1.008	1.011	1.438	1.046	1.000	1.000	1.000	1.000

The nonlinear placer in DrlGoFPGA and DrlGoFPGA 2.0 are selected as DREAMPlaceFPGA.
IOBUF+LFDR+NP: IOBUF placement time + LFDR initial placement time + LFDR initial placement fine-tuning time.

TABLE VI
COMPARISON OF OPENPARF AND DRLGOFPGA 2.0 ON ISPD'2016 BENCHMARKS (HPWL AND RWL IN 10^3, GPRT AND ORT IN SECONDS)

Design	OpenPARF [12]				DrlGoFPGA [15]				DrlGoFPGA 2.0			
	RWL	HPWL	GPRT	ORT	RWL	HPWL	GPRT	ORT	RWL	HPWL	GPRT (IOBUF+LFDR+NP)	ORT
FPGA01	329.7	213.3	15.7	27.1	323.8	208.5	15.7	26.4	323.4	207.9	14.3 (0.6+0.5+13.2)	24.5
FPGA02	693.4	482.5	25.2	42.5	678.2	482.2	22.1	39.1	676.5	469.7	14.6 (0.6+0.5+13.6)	31.1
FPGA03	3003.0	1978.5	25.5	86.6	2958.0	1941.8	26.6	85.4	2896.0	1892.4	17.5 (0.7+0.5+16.3)	78.8
FPGA04	6014.6	3986.2	25.1	78.3	5957.1	3961.6	26.1	78.6	5862.5	3923.5	17.3 (0.7+0.5+16.1)	72.9
FPGA05	12188.6	7951.0	33.2	88.7	11827.5	7715.4	30.1	84.1	11942.1	7849.8	17.2 (0.7+0.5+16.1)	70.8
FPGA06	5922.4	3313.9	28.6	178.6	5853.5	3280.4	30.1	180.0	5675.8	3152.6	19.9 (0.7+0.5+18.7)	146.7
FPGA07	10084.7	6159.6	27.9	150.0	10050.6	6150.9	29.0	150.7	10065.2	6176.9	20.2 (0.7+0.5+19.0)	136.2
FPGA08	9324.9	6554.8	31.8	124.2	9265.7	6455.3	32.3	122.8	9331.3	6497.1	20.8 (0.7+0.5+19.6)	117.3
FPGA09	12540.6	7979.3	31.8	189.6	12563.0	7971.4	32.5	187.7	12530.3	7903.2	23.1 (0.7+0.6+21.8)	162.9
FPGA10	5574.6	3212.9	30.0	179.1	5541.4	3181.1	30.9	182.1	5528.9	3131.5	20.1 (0.7+0.6+18.7)	164.9
FPGA11	13145.4	8923.4	32.6	151.0	12938.3	8846.7	33.7	154.9	12961.7	8875.4	22.4 (0.7+0.5+21.2)	134.4
FPGA12	6961.2	4248.9	33.8	200.8	6802.4	4164.8	34.1	208.6	6856.6	4265.8	28.3 (0.7+0.6+27.0)	202.5
Ratio	1.018	1.018	1.461	1.137	1.004	1.006	1.462	1.125	1.000	1.000	1.000	1.000

The nonlinear placer in DrlGoFPGA and DrlGoFPGA 2.0 are selected as OpenPARF.
IOBUF+LFDR+NP: IOBUF placement time + LFDR initial placement time + LFDR initial placement fine-tuning time.

fine-tuning the LFDR placement models of FPGA01-FPGA12, even when replacing the nonlinear placer with OpenPARF during testing, better GP optimization results can still be obtained, demonstrating that the LFDR placement model has good generalization performance.

E. Evaluation on ISPD'2017 benchmarks

Table VII presents the comparison results of OpenPARF, DrlGoFPGA, and DrlGoFPGA 2.0 on ISPD'2017 benchmarks. As DREAMPlaceFPGA does not support clock routing constraints on ISPD'2017 benchmarks, we do not include it in the comparison. The result of Table VII is summarized as follows:

- In Table VII, compared to OpenPARF and DrlGoFPGA, DrlGoFPGA 2.0 achieved a 27.3% and 27.9% decrease in GPRT, a 5.1% and 5.0% decrease in ORT, a 4.6% and 4.3% decrease in HPWL, and a 2.8% and 2.6% decrease in RWL.

Therefore, when faced with FPGA designs with clock constraints, DrlGoFPGA 2.0 achieved better GP, LG, DP, and routing results.

F. Ablation study

1. To explore the superiority of our proposed MAGTN policy network structure, we replaced the policy network structure with different graph neural networks for experimentation. The specific design is as follows:

1) MAGNN: consists of 3 graph convolution networks (GCNs) [30] and 3 ReLUs [25] activation functions. Apply the results to the ReLU after each GCN. The first, second, and third GCN hidden/output layers have dimensions of 512, 256, and 2, respectively.

2) MAGAT: consists of 3 graph attention networks (GATs) [31] and 3 exponential linear units (ELUs) [32] activation functions. Apply the results to the ELU after each GAT. The first, second, and third GAT hidden/output layers have dimensions of 256, 128, and 2, with 2, 1, and 1 attention heads, respectively.

3) MAGIN: consists of 3 graph isomorphism networks (GINs) [33] and 3 ReLUs activation functions. Apply the results to the ReLU after each GIN. The first, second, and third GIN hidden/output layers have dimensions of 512, 256, and 2, respectively.

Additionally, we conducted ablation studies on GTNs with different TransformerConv depths, including layers 2, 4, and

TABLE VII
COMPARISON OF OPENPARF AND DRLGOFPGA 2.0 ON ISPD'2017 BENCHMARKS (HPWL AND RWL IN 10^3, GPRT AND ORT IN SECONDS)

Design	OpenPARF [12]				DrlGoFPGA [15]				DrlGoFPGA 2.0			
	RWL	HPWL	GPRT	ORT	RWL	HPWL	GPRT	ORT	RWL	HPWL	GPRT (IOBUF+LFDR+NP)	ORT
CLK-FPGA01	2134.2	1602.7	28.6	89.2	2170.1	1626.1	29.3	89.4	2096.7	1479.9	19.4 (0.7+0.5+18.2)	82.5
CLK-FPGA02	2565.6	1727.2	29.4	85.6	2571.6	1730.4	30.2	86.1	2426.4	1576.4	20.2 (0.7+0.5+19.1)	78.0
CLK-FPGA03	6115.3	4410.8	40.4	141.3	6008.4	4339.2	40.3	141.0	5927.8	4258.5	35.1 (0.7+0.5+33.9)	139.3
CLK-FPGA04	4356.2	3022.6	42.7	120.5	4302.4	2998.4	39.6	114.5	4213.7	2931.5	29.2 (0.7+0.5+28.0)	106.2
CLK-FPGA05	5293.8	3750.3	36.6	137.0	5284.5	3723.0	39.1	136.9	5190.3	3641.0	32.1 (0.7+0.6+30.8)	132.6
CLK-FPGA06	6378.5	4531.2	40.4	148.3	6336.1	4532.9	35.2	143.0	6260.5	4456.3	32.0 (0.7+0.6+30.6)	144.3
CLK-FPGA07	2660.7	1849.9	28.5	92.8	2659.1	1836.3	29.6	93.3	2604.5	1798.2	25.5 (0.7+0.5+24.4)	89.0
CLK-FPGA08	2035.7	1498.9	26.8	79.3	2038.2	1496.6	27.6	79.9	1946.3	1361.5	16.3 (0.7+0.5+15.1)	70.6
CLK-FPGA09	2604.2	1833.4	29.7	96.8	2607.6	1830.2	29.8	96.6	2519.7	1666.0	19.7 (0.7+0.5+18.5)	89.8
CLK-FPGA10	4664.1	3214.1	33.8	131.0	4641.7	3194.7	33.3	130.5	4567.6	3156.5	30.4 (0.7+0.5+29.2)	127.4
CLK-FPGA11	4365.9	3037.4	32.6	120.8	4317.0	3010.3	32.3	123.0	4274.8	2977.4	30.3 (0.7+0.6+29.0)	118.7
CLK-FPGA12	3475.3	2365.1	30.2	109.2	3498.7	2388.8	31.5	111.9	3447.5	2359.6	29.5 (0.7+0.5+28.3)	110.2
CLK-FPGA13	4440.9	3103.0	32.8	115.8	4421.9	3100.1	35.5	118.5	4273.7	3015.4	29.5 (0.7+0.5+28.3)	117.0
Ratio	1.028	1.046	1.273	1.051	1.026	1.043	1.279	1.050	1.000	1.000	1.000	1.000

The nonlinear placer in DrlGoFPGA and DrlGoFPGA 2.0 are selected as OpenPARF.
IOBUF+LFDR+NP: IOBUF placement time + LFDR initial placement time + LFDR initial placement fine-tuning time.

TABLE VIII
COMPARISON OF MAGNN, MAGAT, MAGIN, AND MAGTN-2/4/6 ON ISPD'2016 BENCHMARKS (HPWL IN 10^3, GPRT IN SECONDS)

Design	MAGNN		MAGAT		MAGIN		MAGTN-4		MAGTN-6		MAGTN-2 (Ours)	
	HPWL	GPRT	HPWL	GPRT	HPWL	GPRT	HPWL	GPRT	HPWL	GPRT	HPWL	GPRT
FPGA01	190.2	7.07	192.2	8.2	192.4	7.0	187.3	6.6	188.6	6.9	189.2	7.0
FPGA02	628.3	24.92	486.4	21.1	487.2	21.3	476.3	13.3	490.6	16.3	477.6	12.8
FPGA03	2031.1	18.94	2023.6	20.1	1988.0	17.6	1941.7	11.9	1970.3	15.2	1940.0	11.9
FPGA04	4002.0	20.64	4048.9	21.9	4022.9	20.4	3987.5	12.7	3990.3	14.9	3983.3	11.4
FPGA05	8041.9	21.65	8146.7	22.7	8116.3	20.1	7975.9	11.3	8081.3	17.7	7976.2	11.9
FPGA06	3386.0	19.11	3385.4	21.3	3353.4	18.6	3267.7	12.3	3363.4	16.2	3255.9	12.5
FPGA07	6234.9	18.56	6233.5	20.2	6202.7	18.3	6162.1	11.6	6196.0	15.7	6208.7	11.9
FPGA08	6644.5	19.97	6638.3	19.4	6609.2	20.6	6619.5	14.3	6603.9	16.6	6593.9	14.4
FPGA09	8172.0	22.96	8188.4	23.0	8173.6	21.3	8160.6	14.1	8194.7	20.5	8162.9	14.3
FPGA10	3283.9	20.67	3276.5	21.9	3257.8	20.2	3252.4	12.2	3270.5	16.4	3244.2	12.2
FPGA11	8982.2	20.68	8895.5	20.6	8918.2	18.5	9004.0	17.3	9219.2	17.5	8986.4	16.9
FPGA12	4351.0	22.49	4209.3	22.5	4319.4	22.5	4235.5	18.0	4247.9	18.9	4238.9	18.6
Ratio	1.039	1.534	1.013	1.578	1.011	1.461	0.999	0.999	1.011	1.246	1.000	1.000

The nonlinear placer in DrlGoFPGA 2.0 is selected as DREAMPlaceFPGA.

6 (denoted as MAGTN-2/4/6). The MAGTN-2 is used in this work, and the MAGTN-4/6 settings are as follows:

4) MAGTN-4: consists of 1 Embedding layer, 4 TransformerConv [24] and 3 ReLUs activation functions. Except for the last layer, apply the results to ReLU after each TransformerConv. The hidden of the Embedding layer is set to 128, and the heads of the 4 TransformerConv are set to 2, 2, 2, and 1, respectively.

5) MAGTN-6: consists of 1 Embedding layer, 6 TransformerConv and 5 ReLUs activation functions. Except for the last layer, apply the results to ReLU after each TransformerConv. The hidden of the Embedding layer is set to 128, and the heads of the 6 TransformerConv are set to 2, 2, 2, 2, 2, and 1, respectively.

The input node features of MAGNN, MAGAT, and MAGIN are defined as $[\mathbf{0.1}, \mathbf{0.1}]_{n\times 2}$. The pre-training and fine-tuning methods for the LFDR placement model are described in Section IV-C. Table VIII presents the comparison results of MAGNN, MAGAT, MAGIN, and MAGTN-2/4/6 on ISPD'2016 benchmarks. In Table VIII, compared to MAGNN, MAGAT, and MAGIN, MAGTN-2 (Ours) achieved a 53.4%, 57.8%, and 46.1% decrease in GPRT, a 3.9%, 1.3%, and 1.1% decrease in HPWL. Therefore, the proposed MAGTN-2 can achieve better GP results, proving its superiority. Additionally, there is only a 0.1% difference in HPWL and GPRT between MAGTN-4 and MAGTN-2, indicating that the lightweight LFDR LNR graph only requires 2 layers of TransformerConv to achieve sufficient global information exchange, and increasing the number of layers does not bring significant performance gains. MAGTN-6 showed an increase of 1.1% and 24.6% in HPWL and GPRT compared to MAGTN-2. This indicates that excessively deep graph network structures can actually cause over-smoothing or overfitting, which damages the performance of the placement model. As the number of TransformerConv layers increases, the training time of the model will also increase. Therefore, the proposed MAGTN-2 achieves the best balance between efficiency and effectiveness.

2. To further investigate the effect of LFDR initial placement on improving the quality of GP, we removed the influence factor of IOBUF placement optimization in DrlGoFPGA 2.0, and kept its IOBUF placement position consistent with DREAMPlaceFPGA and OpenPARF. Table IX presents the comparison results of DrlGoFPGA 2.0 without IOBUF optimization (denoted as DF 2.0-w/o-IOBUF) with DREAMPlaceFPGA and OpenPARF. Compared to DREAMPlaceFPGA and OpenPARF, DF 2.0-w/o-IOBUF+DREAMPlaceFPGA/OpenPARF

TABLE IX
COMPARISON OF DREAMPLACEFPGA, OPENPARF AND DF 2.0-W/O-IOBUF (HPWL AND RWL IN 10^3, GPRT AND ORT IN SECONDS)

Design	DF 2.0-w/o-IOBUF+DREAMPlaceFPGA				DF 2.0-w/o-IOBUF+OpenPARF				Design	DF 2.0-w/o-IOBUF+OpenPARF			
	RWL	HPWL	GPRT	ORT	RWL	HPWL	GPRT	ORT		RWL	HPWL	GPRT	ORT
FPGA01	338.0	191.5	6.2	23.3	326.1	211.6	13.6	24.7	CLK-FPGA01	2071.9	1457.8	19.1	82.5
FPGA02	691.6	482.2	11.8	47.4	684.4	473.4	16.6	33.9	CLK-FPGA02	2505.8	1684.5	26.4	83.3
FPGA03	3068.5	2007.3	11.5	190.3	3025.9	1966.6	17.5	80.3	CLK-FPGA03	6010.8	4308.5	34.9	141.0
FPGA04	6024.7	4008.3	11.7	187.7	5904.3	3972.4	17.0	73.4	CLK-FPGA04	4230.5	2943.3	29.1	108.5
FPGA05	12237.8	8049.8	11.4	191.6	12133.3	7976.6	16.9	71.1	CLK-FPGA05	5208.3	3675.1	30.8	131.8
FPGA06	5896.5	3279.2	11.7	445.4	5700.2	3170.7	19.4	155.3	CLK-FPGA06	6288.7	4445.6	33.0	146.3
FPGA07	10174.8	6148.3	11.7	449.3	10142.8	6266.8	19.5	146.1	CLK-FPGA07	2592.3	1798.5	25.0	88.3
FPGA08	9544.2	6658.4	13.5	496.4	9326.0	6554.3	21.6	113.1	CLK-FPGA08	1942.7	1363.4	16.3	71.2
FPGA09	12907.6	8181.7	13.1	754.0	12664.8	7978.2	24.4	175.0	CLK-FPGA09	2559.0	1791.0	25.1	92.0
FPGA10	5813.5	3254.0	11.9	676.6	5513.2	3243.4	19.8	169.9	CLK-FPGA10	4588.1	3166.4	29.9	127.8
FPGA11	13039.8	8737.5	16.0	688.7	12965.6	8785.0	22.1	136.7	CLK-FPGA11	4311.3	3002.3	30.0	120.1
FPGA12	7333.9	4202.2	17.3	1109.5	6965.1	4314.9	29.7	211.8	CLK-FPGA12	3409.2	2333.9	29.7	111.8
Ratio	0.988	0.985	0.532	0.960	0.993	0.996	0.703	0.914	CLK-FPGA13	4304.8	3029.3	28.9	116.2
									Ratio	0.977	0.969	0.828	0.965

The nonlinear placer in DF 2.0-w/o-IOBUF+DREAMPlaceFPGA/OpenPARF are selected as DREAMPlaceFPGA/OpenPARF.
Ratio: Calculated from the experimental results of DREAMPlaceFPGA and OpenPARF in Tables V, VI, and VII.

TABLE X
COMPARISON OF DREAMPLACEFPGA, OPENPARF AND DF 2.0-PT (HPWL IN 10^3, GPRT IN SECONDS)

Design	DF 2.0-PT+DREAMPlaceFPGA		DF 2.0-PT+OpenPARF		Design	DF 2.0-PT+OpenPARF	
	HPWL	GPRT	HPWL	GPRT		HPWL	GPRT
FPGA01	189.8	7.0	213.6	9.6	CLK-FPGA01	1480.3	18.5
FPGA02	478.3	10.7	470.1	13.8	CLK-FPGA02	1746.0	27.3
FPGA03	2041.3	15.7	1962.0	22.2	CLK-FPGA03	4390.7	39.0
FPGA04	4033.9	17.3	3989.5	22.0	CLK-FPGA04	2998.6	30.1
FPGA05	8070.6	17.4	7996.3	28.9	CLK-FPGA05	3848.3	34.4
FPGA06	3559.5	19.1	3293.6	26.1	CLK-FPGA06	4511.9	37.0
FPGA07	6152.2	16.1	6125.3	24.7	CLK-FPGA07	1838.8	26.7
FPGA08	6644.3	17.4	6525.9	29.5	CLK-FPGA08	1531.6	23.6
FPGA09	8170.9	20.2	7999.9	29.9	CLK-FPGA09	1843.3	28.8
FPGA10	3314.2	18.2	3234.1	28.1	CLK-FPGA10	3185.8	30.7
FPGA11	8975.7	18.0	8794.5	30.4	CLK-FPGA11	2998.8	30.0
FPGA12	4262.7	19.7	4284.0	37.2	CLK-FPGA12	2347.9	28.8
Ratio	0.997	0.701	0.997	0.868	CLK-FPGA13	3122.8	30.5
					Ratio	0.996	0.892

The nonlinear placer in DF 2.0-PT+DREAMPlaceFPGA/OpenPARF are selected as DREAMPlaceFPGA/OpenPARF.
Ratio: Calculated from the experimental results of DREAMPlaceFPGA and OpenPARF in Tables V, VI, and VII.

reduced RWL by 1.2% and 0.7%, HPWL by 1.5% and 0.4%, GPRT by 46.8% and 29.7%, and ORT by 4.0% and 8.6%, respectively, on the ISPD'2016 benchmarks. Compared to OpenPARF, DF 2.0-w/o-IOBUF+OpenPARF reduced RWL by 2.3%, HPWL by 3.1%, GPRT by 17.2%, and ORT by 3.5% on the ISPD'2017 benchmarks. Therefore, even when IOBUF is considered a fixed instance, the experimental results still demonstrate the effectiveness of the proposed method.

3. To explore the generalization ability of the LFDR placement model, we will directly apply the pre-trained LFDR model on FPGA02 to various designs (denoted as DF 2.0-PT). DF 2.0-PT includes IOBUF optimization. The experimental results are shown in Table X. Compared to DREAMPlaceFPGA and OpenPARF, DF 2.0-PT+DREAMPlaceFPGA/OpenPARF reduced HPWL by 0.3% and 0.3%, and GPRT by 29.9% and 13.2%, respectively, on ISPD'2016 benchmarks. Compared to OpenPARF, DF 2.0-PT+OpenPARF reduced HPWL by 0.4% and GPRT by 10.8% on the ISPD'2017 benchmarks. Additionally, we removed the IOBUF optimization and fixed its positions consistent with DREAMPlaceFPGA and OpenPARF. Then, the pre-trained LFDR model on FPGA02 will be directly applied to various designs (denoted as DF 2.0-PT-w/o-IOBUF). The experimental results are shown in Table XI. Compared to DREAMPlaceFPGA and OpenPARF, DF 2.0-PT-w/o-IOBUF is almost identical to them on HPWL, while reducing by 12.5% $\sim$ 29.9% on GPRT. Therefore, the pre-trained LFDR model has a certain degree of generalization ability, especially in accelerating the GPRT of each design, which also proves that the LFDR model can effectively learn physical constraints, such as density contained in labels during the training process. However, there is still a certain gap between pre-trained models and fine-tuned models on HPWL. In the future, we will focus on researching how to enable pre-trained LFDR models to learn different features in each design to improve generalization ability.

V. CONCLUSION

This work proposes a new FPGA GP framework, DrlGoFPGA 2.0. DrlGoFPGA 2.0 first captures complex topological relationships through MAGTN and generates LFDR initial placement that conforms to circuit connection features. Then, a nonlinear placement is used to locally optimize the initial

TABLE XI
COMPARISON OF DREAMPLACEFPGA, OPENPARF AND DF 2.0-PT-W/O-IOBUF (HPWL IN 10^3, GPRT IN SECONDS)

Design	DF 2.0-PT-w/o-IOBUF+DREAMPlaceFPGA		DF 2.0-PT-w/o-IOBUF+OpenPARF		Design	DF 2.0-PT-w/o-IOBUF+OpenPARF	
	HPWL	GPRT	HPWL	GPRT		HPWL	GPRT
FPGA01	192.7	6.1	211.3	12.7	CLK-FPGA01	1454.8	18.1
FPGA02	493.4	14.9	471.3	13.4	CLK-FPGA02	1729.7	27.6
FPGA03	2043.5	16.0	1968.2	21.9	CLK-FPGA03	4432.1	37.8
FPGA04	4055.9	17.8	4002.7	21.6	CLK-FPGA04	3113.1	28.2
FPGA05	8067.4	18.2	7950.7	25.7	CLK-FPGA05	3761.9	34.5
FPGA06	3360.3	17.0	3320.6	25.1	CLK-FPGA06	4521.0	36.1
FPGA07	6255.2	16.5	6161.6	24.0	CLK-FPGA07	1880.9	26.4
FPGA08	6646.1	17.3	6548.0	28.6	CLK-FPGA08	1525.7	24.0
FPGA09	8205.3	19.9	8030.8	29.2	CLK-FPGA09	1828.0	27.7
FPGA10	3347.0	18.3	3255.4	27.6	CLK-FPGA10	3244.8	30.2
FPGA11	8931.5	15.4	8825.0	29.6	CLK-FPGA11	3047.7	29.1
FPGA12	4327.0	20.7	4304.7	36.6	CLK-FPGA12	2372.1	27.9
Ratio	1.000	0.701	0.999	0.859	CLK-FPGA13	3102.2	29.9
					Ratio	0.999	0.875

The nonlinear placer in DF 2.0-PT-w/o-IOBUF+DREAMPlaceFPGA/OpenPARF are selected as DREAMPlaceFPGA/OpenPARF.
Ratio: Calculated from the experimental results of DREAMPlaceFPGA and OpenPARF in Tables V, VI, and VII.

placement of LFDR based on physical constraints. The experimental results of ISPD'2016/2017 benchmarks show that compared to the SOTA nonlinear placers DREAMPlaceFPGA, OpenPARF, and DrlGoFPGA, DrlGoFPGA 2.0 can achieve faster GP speeds, as well as better GP HPWL and routed wirelength. After training and fine-tuning LFDR placement models on FPGA01-FPGA12 using DrlGoFPGA with DREAMPlaceFPGA as the nonlinear placer for label generation, the models still achieve better GP optimization results even when OpenPARF replaces the nonlinear placer during testing, demonstrating strong generalization. We conducted ablation studies on the effects of different types of graph neural networks and without IOBUF placement optimization on GP results, verifying the superiority of the proposed MAGTN.

We believe that DrlGoFPGA 2.0 offers a new paradigm for designing high-performance FPGA physical implementation tools. Future work will focus on quickly migrating DrlGoFPGA 2.0 to different FPGA architecture definition files to improve its versatility on academic and industrial benchmarks, and on enhancing the generalization of pre-trained models.

ACKNOWLEDGMENTS

The author used ChatGPT for grammar checking in the manuscript.

REFERENCES

[1] Y. Meng and J. Yang, "Research on heterogeneous acceleration platform based on FPGA," in *ITM Web of Conferences*, volume 45, pages 01029, EDP Sciences, 2022.

[2] J. Mai, Y. Meng, Z. Di, and Y. Lin, "Multi-electrostatic FPGA placement considering SLICEL-SLICEM heterogeneity and clock feasibility," in *Proc. DAC*, 2022, pp. 649-654.

[3] N. Zhang, X. Chen, and N. Kapre, "Rapidlayout: Fast hard block placement of fpga-optimized systolic arrays using evolutionary algorithm," *ACM Transactions on Reconfigurable Technology and Systems (TRETS)*, 15(4):1-23, 2022.

[4] G. Chen and J. Cong, "Simultaneous placement with clustering and duplication," in *Proceedings of the 41st annual Design Automation Conference*, pages 740-772, 2004.

[5] K. E. Murray, O. Petelin, S. Zhong, J. M. Wang, M. Eldafrawy, J.-P. Legault, E. Sha, A. G. Graham, J. Wu, M. J. P. Walker, H. Zeng, P. Patros, J. Luu, K. B. Kent, and V. Betz, "VTR 8: High-performance CAD and customizable FPGA architecture modelling," *ACM TRETS*, vol. 13, no. 2, pp. 1–55, Jun. 2020.

[6] C.-W. Pui, G. Chen, W.-K Chow, K.-C Lam, J Kuang, P. Tu, H. Zhang and E. F. Y. Young, and B. Yu, "RippleFPGA: A routability-driven placement for large-scale heterogeneous FPGAs," in *Proc. ICCAD*, 2016, pp. 1-8.

[7] R. Pattison, Z. Abuowaimer, S. Areibi, G. Gréwal, and A. Vannelli, "GPlace: A congestion-aware placement tool for ultrascale FPGAs," in *Proc. ICCAD*, 2016, pp. 1-7.

[8] T. Liang, G. Chen, J. Zhao, S. Sinha, and W. Zhang, "AMF-placer: High-performance analytical mixed-size placer for FPGA," in *Proc. ICCAD*, 2021, pp. 1-9.

[9] W. Li, Y. Lin, and D. Z. Pan, "elfPlace: Electrostatics-based placement for large-scale heterogeneous FPGAs," in *Proc. ICCAD*, 2019, pp. 1-8.

[10] R. S. Rajarathnam, M. B. Alawieh, Z. Jiang, M. Iyer, and D. Z. Pan, "DREAMPlaceFPGA: An open-source analytical placer for large scale heterogeneous FPGAs using deep-learning toolkit," in *Proc. ASP-DAC*, 2022, PP. 300-306.

[11] R. S. Rajarathnam, Z. Jiang, M. A. Iyer, and D. Z. Pan, "Dreamplacefpga-pl: An open-source gpu-accelerated packer-legalizer for heterogeneous fpgas," *Proceedings of the 2023 International Symposium on Physical Design*, 2023, pp. 175-184.

[12] J. Mai, J. Wang, Z. Di, G. Luo, Y. Liang, and Y. Lin, "OpenPARF: An open-source placement and routing framework for large-scale heterogeneous FPGAs with deep learning toolkit," in *Proc. ASICON*, 2023, pp. 1-4.

[13] J. Lu, P. Chen, C.-C Chang, L. Sha, D. J.-H. Huang, C.-C. Teng, and C.-K. Cheng, "ePlace: Electrostatics-based placement using fast fourier transform and Nesterov's method," *ACM Transactions on Design Automation of Electronic Systems (TODAES)*, 20(2):1-34, 2015.

[14] A. Paszke, "Pytorch: An imperative style, high-performance deep learning library," *arXiv preprint arXiv:1912.01703*, 2019.

[15] K. Yang, J. Zhai, L. Xiang, Z. Huang, D. Wu, Y. Wang, K. Zhao, M. Lei, and Z. He, "DrlGoFPGA: FPGA Global Placement Considering Input-Output Buffer Based on Deep Reinforcement Learning and Gradient Optimization," *IEEE Transactions on Circuits and Systems I: Regular Papers*, 2025.

[16] Xilinx, "UltraScale Architecture and Product Overview," Accessed: Apr. 29, 2024. [Online]. Available: http://www.xilinx.com/support/documentation/data_sheets/ds890-ultrascale-overview.pdf

[17] I. Bustany, G. Gasparyan, A. Gupta, A. B. Kahng, M. Kalase, W. Li, and B. Pramanik, "The 2023 mlcad fpga macro placement benchmark design suite and contest results," *2023 ACM/IEEE 5th Workshop on Machine Learning for CAD (MLCAD)*, Snowbird, UT, USA, 2023, pp. 1-6.

[18] Q. Xu, H. Geng, S. Chen, B. Yuan, C. Zhuo, Y. Kang, and X. Wen, "GoodFloorplan: Graph convolutional network and reinforcement learning-based floorplanning," *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, 41(10):3492-3502, 2021.

[19] Y. Liu, Z. Ju, Z. Li, M. Dong, H. Zhou, J. Wang F. Yang, X. Zeng, and L. Shang, "Floorplanning with graph attention," in *Proc. DAC*, 2022, pp. 1303-1308.
[20] M. Amini, Z. Zhang, S. Penmetsa, Y. Zhang, J. Hao, and W. Liu, "Generalizable floorplanner through corner block list representation and hypergraph embedding," *Proceedings of the 28th ACM SIGKDD Conference on Knowledge Discovery and Data Mining*, 2022, pp. 2692-2702.
[21] D. U. Lim, and H. Park, "Graph Neural Network-Based Detailed Placement Optimization Framework," in *Proc. ISQED*, 2024, pp. 1-6.
[22] S. Yun, M. Jeong, R. Kim, J. Kang, and H. J. Kim, "Graph transformer networks," *Advances in neural information processing systems*, volume 32, 2019.
[23] M.-K. Hsu, Y.-W. Chang, and V. Balabanov, "TSV-aware analytical placement for 3D IC designs," in *Proc. DAC*, 2011, pp. 664-669.
[24] Y. Shi, Z. Huang, S. Feng, H. Zhong, W. Wang, and Y. Sun, "Masked label prediction: Unified message passing model for semi-supervised classification," *arXiv preprint arXiv:2009.03509*, 2020.
[25] V. Nair and G. E. Hinton, "Rectified Linear Units Improve Restricted Boltzmann Machines Vinod Nair," *Omnipress*, 2010.
[26] I. Loshchilov and F. Hutter, "Decoupled weight decay regularization," *arXiv preprint arXiv:1711.05101*, 2017.
[27] A. Paszke, S. Gross, S. Chintala, G. Chanan, E. Yang, Z. DeVito, Z. Lin, A. Desmaison, L. Antiga, and A. Lerer, "Automatic differentiation in pytorch," 2017.
[28] S. Yang, A. Gayasen, C. Mulpuri, S. Reddy, and R. Aggarwal, "Routability-driven FPGA placement contest," *Proceedings of the 2016 on International Symposium on Physical Design*, 2016, pp. 139-143.
[29] S. Yang, C. Mulpuri, S. Reddy, M. Kalase, S. Dasasathyan, M. E. Dehkordi, M. Tom, and R. Aggarwal, "Clock-aware FPGA placement contest," *Proceedings of the 2017 ACM on International Symposium on Physical Design*, 2017, pp. 159-164.
[30] T. N. Kipf and M. Welling, "Semi-supervised classification with graph convolutional networks," *arXiv preprint arXiv:1609.02907*, 2016.
[31] P. Velickovic, G. Cucurull, A. Casanova, A. Romero, P. Lio, and Y. Bengio, "Graph attention networks," *stat*, 1050(20):10-48850, 2017.
[32] D.-A. Clevert, T. Unterthiner, and S. Hochreiter, "Fast and accurate deep network learning by exponential linear units (elus)," *arXiv preprint arXiv:1511.07289*, 2015.
[33] K. Xu, W. Hu, J. Leskovec, and S. Jegelka, "How powerful are graph neural networks?," *arXiv preprint arXiv:1810.00826*, 2018.
[34] P. Sedgwick, "Pearson's correlation coefficient," *Bmj*, 345, 2012.
[35] S. Ji, Z. Zhang, S. Ying, L. Wang, X. Zhao, and Y. Gao, "Kullback–Leibler divergence metric learning," *IEEE transactions on cybernetics*, 52(4):2047-2058, 2020.

Kang Yang received the B.S. degree in electrical engineering and automation from Guangdong Polytechnic Normal University, Guangdong, China, in 2019 and is currently pursuing the Ph.D. degree with the School of Integrated Circuits, Beijing University of Posts and Telecommunications, Beijing, China. His research interests include AI for EDA, microgrid energy dispatch technology, and bionic robotic arms. He is currently researching FPGA placement and routing algorithms.

Jianwang Zhai is currently an Associate Professor in the School of Integrated Circuits, Beijing University of Posts and Telecommunications, Beijing. He received the B.E. degree in communication engineering from Beijing Jiaotong University, Beijing, China, in 2018, and the Ph.D. degree in computer science and technology from Tsinghua University, Beijing, in 2023. His research interests include architecture modeling, design space exploration, and physical design. He received the William J. McCalla Best Paper Award from ICCAD 2021 and the Best Paper Award Nominations from ASP-DAC 2023 & GLSVLSI 2025.

Liuyu Xiang is currently an Associate Professor with the School of Artificial Intelligence, Beijing University of Posts and Telecommunications. He received his B.Sc. degree from EE, University of Science and Technology of China in 2017 and Ph.D. degree from School of Software, Tsinghua University. His research interests include computer vision and reinforcement learning.

Kang Zhao received the Ph.D. degree in the department of computer science and technology from Tsinghua University, China in 2009. Then, he worked in Tsinghua University, Intel, Xilinx and AMD respectively. When working in Xilinx and AMD, he played as the senior staff manager and leader of the Vitis HLS product team. Now he is the professor of Beijing University of Posts and Telecommunications, and lead the EDA team. His research interests include FPGA and EDA, especially on high-level synthesis, logic synthesis, compiler techniques, placement and floorplan, and other VLSI CAD algorithms.

Wei Li received the bachelor's degree in engineering from the Department of Electronic Engineering and Information Science, University of Science and Technology of China, Anhui, China, in 2005, and the PhD degree from the Institute of Electronics, Chinese Academy of Sciences, Beijing, China, in 2010, with the guidance from prof. Haigang Yang. She is currently an associate professor at the Institute of Computing Technology, Chinese Academy of Sciences, Beijing, China.

Ming Lei received his M.D. at State Key Laboratory of Materials Composites and Advanced Technology, Wuhan University of Technology. Then, he received his Ph.D. from the Laboratory of Nanophysics and Devices, Institute of Physics, Chinese Academy of Sciences. He worked as a postdoctoral fellow at The Hong Kong University of Science and Technology (2007-2008) and the Chinese University of Hong Kong (2009-2010). He is now a professor of the School of Integrated Circuits, Beijing University of Posts and Telecommunications. He has long been engaged in Science and Engineering of Integrated Circuits.

Zhaofeng He received PhD degree in Pattern Recognition and Intelligent System from the Institute of Automation, Chinese Academy of Sciences in 2010. Now he serves as a Professor in Beijing University of Posts and Telecommunications, and is the founder of the Laboratory of Visual Computing and Intelligent System (VCIS). He has authored several top and international conference and journal papers in TPAMI, CVPR, etc. His research interests include biometrics, computer vison, intelligent system, EDA.