DrlGoFPGA: FPGA Global Placement Considering Input-Output Buffer Based on Deep Reinforcement Learning and Gradient Optimization

Kang Yang, Jianwang Zhai, Liuyu Xiang, Zixi Huang, Dan Wu, Yida Wang, Kang Zhao, Ming Lei, and Zhaofeng He

***Abstract*—The placement of the input-output buffer (IOBUF) can impact the performance and power consumption of the FPGA. The existing global placement (GP) methods lack consideration for IOBUF, resulting in a decrease in placement and routing quality. To address this issue, we propose a GP framework, DrlGoFPGA, which combines IOBUF placement based on deep reinforcement learning (DRL) with other instances placement based on gradient optimization (GO). A policy network structure with multi-action sampling is designed to accelerate the running speed of DRL, and a parallelizable reward function is designed to optimize each IOBUF placement action and avoid sparse reward problems. Then, an IOBUF line-network relationship (ILNR) graph creation method is designed to improve the agent's ability to explore optimal solutions, and the graph features of ILNR by capturing them through a graph neural network embedded in the convolutional neural network. Finally, an IOBUF placement legalization method is designed to ensure that the IOBUF position meets the FPGA architecture. The experimental results show that compared with the state-of-the-art placement tools based on GO, DrlGoFPGA can improve GP speed by 13.2%-7×, half-perimeter wirelength by 0.2%-2.6%, and wirelength by 0.2%-1.5% and the IOBUF placement model has good generalization.**

***Index Terms*—FPGA global placement, deep reinforcement learning, gradient optimization, input-output buffer, legalization.**

I. Introduction

MODERN File Programmable Gate Arrays (FPGAs) have become an important hardware platform for digital system design, playing an irreplaceable role in fields such as the automotive industry and artificial intelligence [1].

Received 13 January 2025; revised 16 April 2025; accepted 30 April 2025. This work was supported in part by the National Natural Science Foundation of China under Grant 62176025, Grant 62301066, Grant U21B2045, and Grant 62206012; and in part by the Fundamental Research Funds for the Central Universities under Grant 2023RC72. This article was recommended by Associate Editor J. Zhang. *(Corresponding authors: Zhaofeng He; Jianwang Zhai; Liuyu Xiang.)*

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, Zixi Huang, Dan Wu, Yida Wang, 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; huangzixi@bupt.edu.cn; wudan@bupt.edu.cn; wangyida@bupt.edu.cn; Zhaofenghe@bupt.edu.cn).

This article has supplementary downloadable material available at https://doi.org/10.1109/TCSI.2025.3570805, provided by the authors.

Digital Object Identifier 10.1109/TCSI.2025.3570805

FPGA placement is the mapping of logical circuits onto the physical resources, with the goal of maximizing FPGA performance while minimizing power consumption and area. FPGA placement typically includes GP, legalization (LG), and detailed placement (DP). The GP has a significant effect on the performance of the FPGA [2]. However, GP requires most of the runtime to achieve a good placement result.

Some studies [2], [3], [4], [5], [6] focused on GP acceleration. The open-source placement tools DREAMPlaceFPGA [2] and OpenPARF [4] both utilize the PyTorch deep-learning toolkit and GPU to accelerate the GP speed. The *elfPlace* [3] also utilizes GPU acceleration technology. Shounak et al. utilize the OpenCL and FPGA's ability to support deep pipelines to achieve GP acceleration [5] and propose a parallel diffusion algorithm to accelerate optimization speed and a flow correction algorithm to eliminate possible loops in discrete processes to achieve GP acceleration [6]. Al-Hyari et al. [10] use the convolutional encoder-decoder to predict congestion in FPGA analytical placement iterations and use congestion information to improve decision-making, reducing the placer runtime by 27%-40%. These works greatly shorten placement time while ensuring performance comparable to traditional methods. While shortening the GP time is important, improving the performance of the FPGA placement is more important.

Some studies use deep learning (DL) and reinforcement learning (RL) to assist traditional methods in achieving better placement performance. Elgamma [7] and Murray et al. [8] use directional movement and RL to improve placement solutions based on simulated annealing. Wang et al. [9] introduce the convolutional neural network (CNN) into FPGA analytical placement method, using CNN and adaptive strategies to construct a density framework to improve placement solution. The existing FPGA GP methods treat IOBUF as fixed instances to reduce the placement complexity. However, the IOBUF placement has a significant effect on improving placement performance. In addition, the above work using DL or RL to assist traditional methods requires online learning of new circuits from scratch to enable the agent to adapt to the characteristics of the circuit, resulting in expensive learning time costs.

With the development of artificial intelligence technology, some studies [11], [13] have started using DRL for end-to-end placement learning of macro cells on ASICs, significantly improving placement speed and quality. For placement models that perform poorly on some new circuits, fine-tuning the model can also achieve better placement results than traditional methods. This method is more advantageous for agents to learn more general placement rules to improve model generalization and avoid the expensive learning time cost caused by the need to learn from scratch for new circuits. Directly learning the placement model of macro cells through DRL provides a new approach for optimizing chip placement.

Accordingly, we propose DrlGoFPGA, a FPGA GP framework based on DRL and GO, aimed at improving FPGA placement performance through IOBUF placement optimization. Our key contributions are as follows:

- We propose a GP framework, DrlGoFPGA, which combines IOBUF placement based on DRL with other instance placement based on GO for joint learning.
- We design a policy network structure with multi-action sampling consisting of a graph neural network (GNN) and a CNN to accelerate the running speed of DRL.
- We design a parallelizable reward function to optimize each IOBUF placement action in parallel and avoid the sparse reward problems of DRL.
- We design an ILNR graph creation algorithm based on the IOBUF netlists to improve the agent's ability to explore optimal solutions and embed ILNR knowledge into the CNN of DRL by capturing it through GNN.
- We design an IOBUF placement legalization (IPL) method to quickly avoid IOBUF overlap and ensure that the placement position of IOBUF meets FPGA architecture constraints in DRL training and testing.

This paper is organized as follows: Section II introduces the preliminaries. Section III describes the proposed FPGA GP framework. The experiment results and ablation study are reported in Section IV. Section V summarizes this work.

II. Preliminaries

This section briefly introduces the heterogeneous FPGA architecture, two GP methods, and the GNN used in this work.

A. FPGA Architecture

This work adopts the Xilinx UltraScale architecture of heterogeneous FPGAs, which contains different logic resource types in each column [16]. In Fig. 1, the input-output (I/O), physical layer (PHY) blocks, and the configuration resource (CR) are variably arranged across the FPGA fabric. CR consists of the configurable logic block (CLB), digital signal processor (DSP), and random-access memory (RAM). A CLB consists of 8 basic logic elements (BLEs), with each BLE consisting of 2 look-up tables (LUTs) and 2 flip-flops (FFs). PHY contains input BUF (IBUF) and output BUF (OBUF). IBUF is used to receive signals from external signal sources. It is usually used to protect the internal circuits from noise and interference from external signal sources. OBUF is used to drive the output signals of internal logic circuits to external devices. It is usually used to ensure the quality and reliability of the output signal. IOBUFs are placed in a high-performance input-output bank (HPIOB) or high-range input-output bank (HRIO), as shown in Fig. 2.

The HPIOB or HRIO is distributed at site coordinates $[x \in [66, 103], \quad y \in [0, 30, \cdots, 450]_{1\times 16}$, BEL

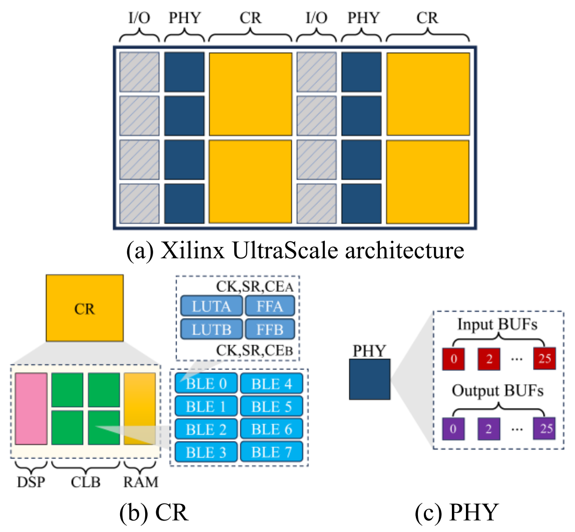


Fig. 1. Composition of Xilinx UltraScale architecture [16].

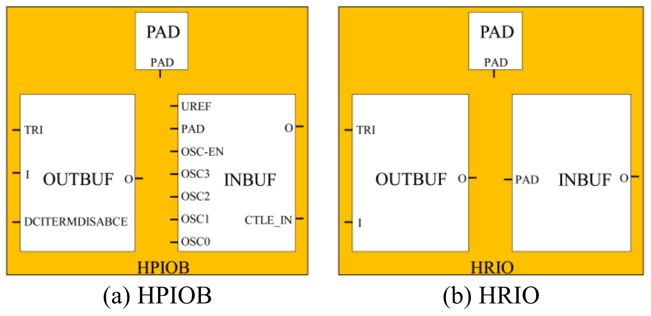


Fig. 2. HPIOB and HRIO architecture.

$z \in [0, 1, \cdots, 25]_{1\times 26}]$ in the Xilinx UltraScale architecture. In the GP results, the three-dimensional coordinates $[x, y, z]$ of HPIOB or HRIO are usually used to represent the placement position of IOBUF. However, in the Xilinx UltraScale architecture, the actual y-index (denoted as y') of the sites for the IOBUF is $y' = (y+z) \in [0, 1, \cdots, 25, 30, \cdots, 475]_{1\times 416}$. To better improve the performance and reliability of input and output signals, HPIOB or HRIO usually only allows the placement of one IBUF or OBUF.

B. Gradient Optimization for FPGA Global Placement

The *elfplace* [3], as one of the state-of-the-art GO method, is used for FPGA GP. The *elfPlace* adopts an electrostatic system representation for GP and uses a weighted-average (WA) model $\tilde{W}(x, y)$ [17] and electrical potential energy Φ_s to approximate the half-perimeter wirelength (HPWL) and density, respectively. The objective function of *elfplace* is shown in (1). DREAMPlaceFPGA [2] is suitable for *elfPlace* on the PyTorch deep-learning toolkit [18], which is developed from DREAMPlace [19] used for ASIC GP.

$$\min_{x,y} f(x, y) = \tilde{W}(x, y) + \sum\nolimits_{s\in S} \lambda_s \left[\Phi_s(x, y) + \frac{c_s}{2}\Phi_s(x, y)^2\right] \tag{1}$$

$$\tilde{W}(x, y) = \sum\nolimits_{e\in E} (\tilde{W}_{e_x} + \tilde{W}_{e_y}) \tag{2}$$

$$\tilde{W}_{e_x} = \frac{\sum_{i\in e} x_i \exp(x_i/\gamma)}{\sum_{i\in e} \exp(x_i/\gamma)} - \frac{\sum_{i\in e} x_i \exp(-x_i/\gamma)}{\sum_{i\in e} \exp(-x_i/\gamma)} \tag{3}$$

$$W(x, y) = \sum_{e \in E} W_e(x, y) = \sum_{e \in E} \left(\max_{i,j \in E} |x_i - x_j| + \max_{i,j \in E} |y_i - y_j| \right) \quad (4)$$

where λ_s is the density multiplier. $\tilde{W}_{e_x}$ is the x-directed WA wirelength model. γ controls the accuracy and smoothness of the HPWL approximation. c_s is used to weight the quadratic penalty term $\Phi_s(x, y)^2$. $W_e(x, y)$ is the HPWL of net e within the set of nets E in the design. $W(x, y)$ is the HPWL of FPGA GP. $s \in S = \{\text{LUT, FF, DSP, RAM}\}$.

C. DRL for ASIC Global Placement

Many studies use DRL to place macro cells on ASIC and significantly improve GP performance. Google [11] proposes a DRL method for chip floorplanning, which can generate chip floorplanning in less than 6 hours. It has better power consumption, performance, and area metrics than or comparable to manually made chips. Cheng et al. [12] propose a joint learning method for ASIC, DeepPlace, which combines macro cell placement based on RL with standard cell placement based on DREAMPlace [19], greatly reducing placement time and wirelength. To overcome the sparse reward problem, DeepPlace employs random network distillation (RND) [20], which has been shown to significantly improve the performance. Lai et al. [13] propose a macro cell placement method based on RL, MaskPlace, which learns position, wirelength, and view information based on rich visual representation to help models take action. This work is better than existing RL methods in terms of wirelength, congestion, and density.

Motivated by the above work, this paper uses DRL to optimize the placement of IOBUF in the FPGA.

D. GNN for DRL

In terms of policy network structure, references [11], [12] both used GNN to capture node embedding knowledge of netlists, effectively improving the agent's ability to explore optimal solutions. In heterogeneous system design, the self-optimizing and self-programming computing system framework [14] uses neural network (NN) to identify features of instruction dependency graphs and distribute tasks via RL, improving performance by 4.12×. The graph analytics based high level synthesis framework [15] constructs dependency graphs to enhance performance by 14.27× through subgraph detection and feature extraction. Yao et al. develop the plasticity on chip framework [21] to optimize chip design by converting high-level programs into weighted directed acyclic graphs. Using GNNs for feature extraction in distributed RL, they achieved a 7.61× performance improvement over traditional network-on-chip methods. Their programmable graph representation learning framework [22] creates dynamic execution graphs, clusters them into software kernels, and employs GNNs to match each kernel with the optimal hardware device, enhancing optimization speed by 2.02-6.42×. These advancements highlight the effectiveness of graph feature extraction and GNNs in improving chip design quality. Motivated by the above work, this paper also applies GNN to the policy network structure design of DRL and uses it to obtain graph features of ILNR.

E. Problem Statement for FPGA Placement

The better placement of IOBUFs can improve the performance, power consumption, and reliability of the FPGA circuit. However, FPGA performance bottlenecks mainly focus on CLB, DSP, and RAM placements with a large number of instances or pins. Therefore, existing placement methods prioritize factors that have a greater impact on FPGA performance, while placement optimization for IOBUF instances is relatively overlooked. If the existing placement methods are directly incorporated into IOBUF instance optimization, the algorithm will need to add more constraints and variables, which may make algorithm design and implementation more difficult, and the computation time will also significantly increase. With the development of FPGA technology and the increasing demand for applications, it will become necessary to design an efficient and reliable IOBUF instance optimization method to improve the overall performance of the FPGA.

This paper aims to improve the quality of FPGA placement and routing by optimizing the IOBUF placement. The optimization objective is to minimize HPWL, i.e. min $W(x, y)$, while ensuring that the GP time is better or equivalent to the state-of-the-art GO method.

III. FPGA Global Placement Framework

In this section, we introduce the FPGA GP flow of DrlGoFPGA in Part A. Parts B, C, D, and E, respectively, list the proposed IOBUF placement methods based on DRL.

A. Global Placement Framework of DrlGoFPGA

In FPGA, the IOBUF instance is used to transmit signals between different logic circuits and provide signal-driving capabilities [23]. The placement of IOBUFs can improve the performance and power consumption of the FPGA. However, the state-of-the-art placers based on GO (e.g., DREAMPlaceFPGA [2] and OpenPARF [4]) do not consider the effect of IOBUF placement on the final placement result. In this work, we propose a GP framework, DrlGoFPGA, which consists of the IOBUF placement based on DRL and the CLB, DSP, and RAM (CDR) placement based on GO. We formulate the IOBUF placement problem as a sequential Markov decision process (MDP). Our MDP consists of three key elements:

1) State s_t: The state comprises of IOBUF placement canvas and the ILNR graph, which provides detailed information about the connection relationship between IOBUFs. We set the size of the IOBUF canvas to 2×416, and defined the mapping relationship between canvas coordinates (x, y) and IOBUF placement coordination in Xilinx UltraScale architecture: $f_x : [0, 1] \rightarrow [66, 103]$, $f_y : [0, 1, \cdots, 415]_{1\times416} \rightarrow [0, 1, \cdots, 25, 30, \cdots, 475]_{1\times416}$. We represent the IOBUF canvas as a binary image, denoted as I.

2) Action a_t: The action space consists of available positions on the 2×416 IOBUF canvas at time t. When the current IOBUF selects a position(x, y), we designate $I_{xy} = 1$.

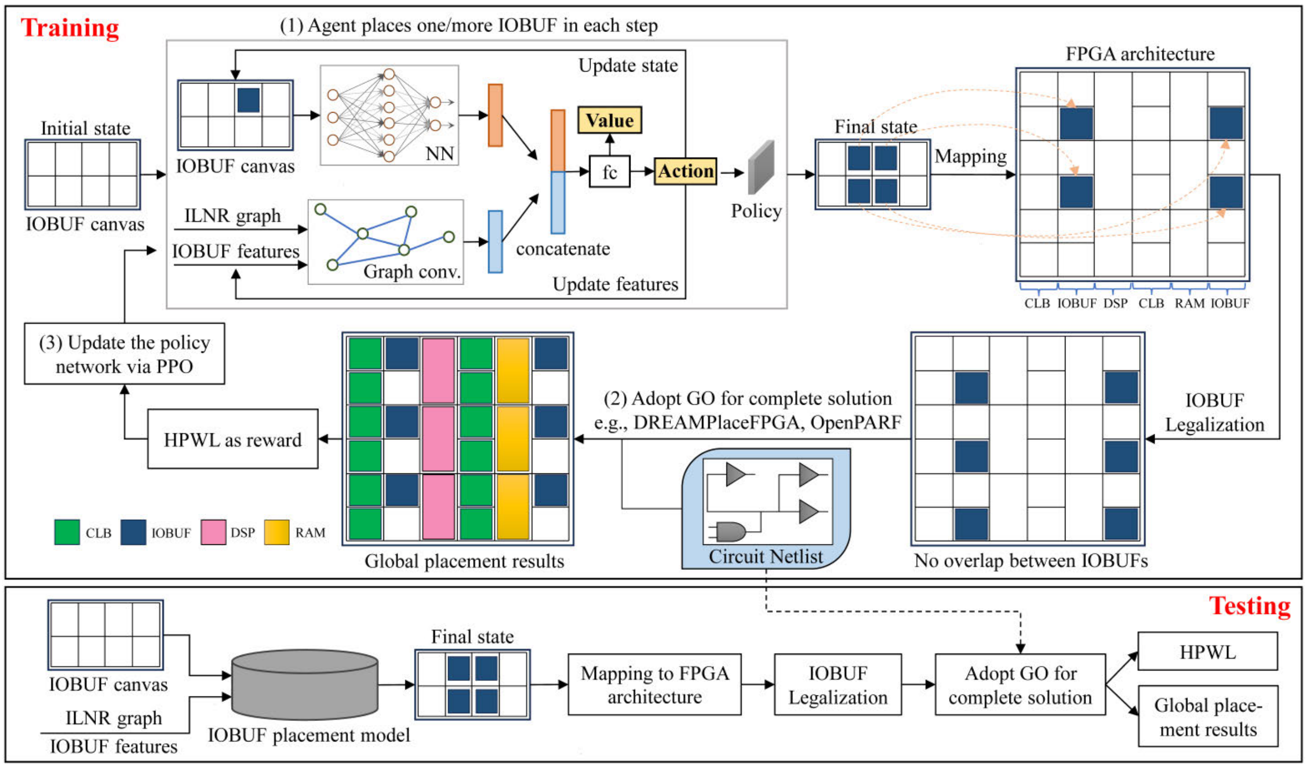


Fig. 3. The global placement flow of DrlGoFPGA.

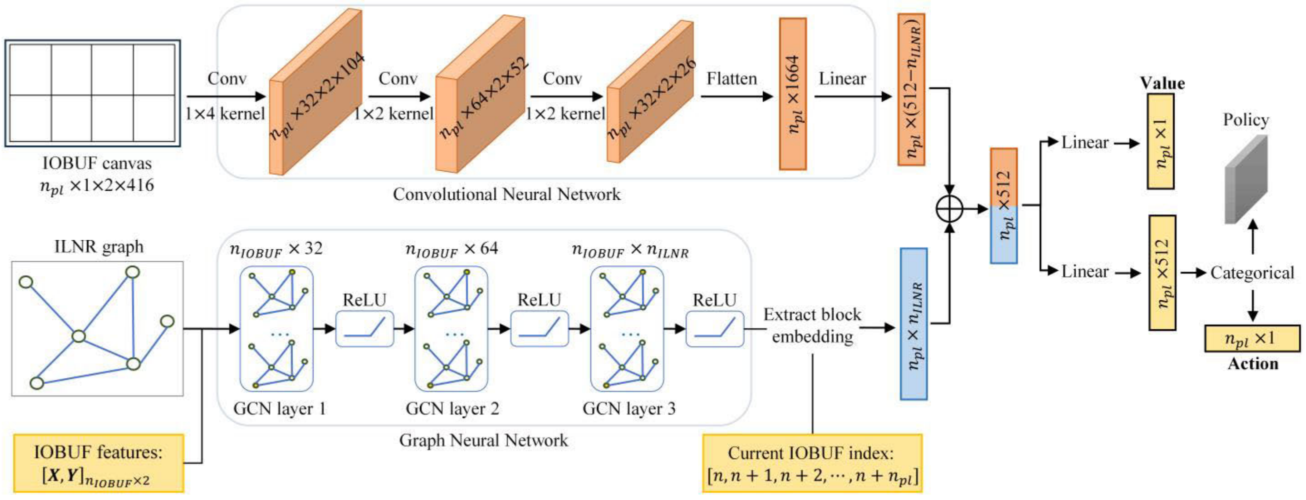


Fig. 4. The policy network structure with multi-action sampling.

3) Reward r_t: The reward is defined as a parallelizable reward function related to HPWL (see Part D of Section III).

The GP flow of DrlGoFPGA is shown in Fig. 3. In the training phase, our RL agent first sequentially maps IOBUFs to valid positions on the IOBUF canvas. The IOBUF canvas serves as the state input for the NN, while the ILNR graph and IOBUF features serve as inputs for the graph convolutional network (GCN). The features of NN and GCN are merged through concatenation, and the actions and values are obtained through a fully connected layer post-sampling. Actions are used to update the state and IOBUF features of each time step. Once all IOBUFs have been placed, we will fix their positions and map them to the FPGA architecture through mapping relationships f_x and f_y. To avoid overlapping, legitimize the placement position of IOBUFs. Then use the placement position of IOBUF, the FPGA architecture, and the complete circuit netlist as inputs for GO to obtain a complete GP solution. Finally, the FPGA GP evaluation metrics HPWL are used as the reward for DRL. The Proximal Policy Optimization (PPO) algorithm [24] is used to update the policy network. In the testing phase, the IOBUF placement model is directly used for placement, and then the positions of IOBUF are mapped and legalized. The GO is used to obtain the GP results and HPWL.

B. Policy Network Structure With Multi-Action Sampling

The traditional policy network structure of DRL samples only one action at each time step, resulting in a slow running speed. To overcome this issue, we have designed a policy network structure with multi-action sampling, as shown in Fig. 4. The IOBUF canvas information is processed through the CNN to obtain global information. The CNN consists of three convolutional layers and one linear layer. The ILNR graph and IOBUF features are handled by the GNN to obtain detailed node information. The IOBUF features are composed of the placement coordinates $(x,\ y)$ of IOBUF on the canvas, and the initial IOBUF features are $[\boldsymbol{X}, \boldsymbol{Y}]_{n_{IOBUF}\times 2} = \mathbf{0}_{n_{IOBUF}\times 2}$($n_{IOBUF}$ is the total IOBUF number). The GNN consists of three GCN [25]. To achieve multi-action sampling at each time step, we define the size of the IOBUF canvas as $n_{pl} \times 1 \times 2 \times 416$ (n_{pl} is the number of parallelization, which is the number of sampling actions). We fuse the $n_{pl} \times (512 - n_{ILNR})$ global information (n_{ILNR} is the dimension of embedding ILNR knowledge into CNN) and $n_{pl} \times n_{ILNR}$ detailed node information obtained based on the current IOBUF index $[n, n+1, n+2, \cdots, n+n_{pl}]$ via concatenation and transmit the $n_{pl} \times 512$ results to the fully connected layer to generate the $n_{pl} \times 512$ probability of actions and $n_{pl} \times 1$ values. Then, the $n_{pl} \times 1$ actions are obtained after sampling, where the initial value of n is 0. After completing each multi-action sampling, update $n \leftarrow n + n_{pl}$, that is, update the placement coordinates of n_{pl} IOBUFs at each time step. One episode is reduced from the original n_{IOBUF} time steps to $n_{IOBUF}//n_{pl}$ time steps. Therefore, the policy network structure with multi-action sampling can accelerate the training and testing speed of DRL. Each n_{pl} IOBUF features are updated according to (5) and (6).

$$\boldsymbol{X}[n, n+1, \cdots, n+n_{pl}]_{n_{pl}\times 1} \leftarrow \boldsymbol{Action}_{n_{pl}\times 1}//416 \quad (5)$$
$$\boldsymbol{Y}[n, n+1, \cdots, n+n_{pl}]_{n_{pl}\times 1} \leftarrow \boldsymbol{Action}_{n_{pl}\times 1}\%416 \quad (6)$$

where $\boldsymbol{X}$ and $\boldsymbol{Y}$ are the horizontal and vertical coordinates of the placement of IOBUF. //represents integer division operation. % represents modulo operation.

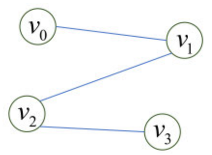
PPO is used to update the policy by optimizing the clipped objective function, as shown in (7).

$$L^{clip}(\theta) = \hat{E}_t\left[\min\left(p_t(\theta)\cdot\hat{A}_t, clip\left(p_t(\theta), 1-\epsilon, 1+\epsilon\right)\hat{A}_t\right)\right] \quad (7)$$

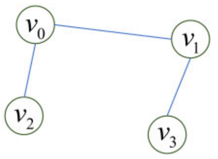
where p_t is the probability ratio of the new policy to the old policy and $\hat{A}_t$is the estimated advantage at time step t.

C. IOBUF Line-Network Relationship Graph Creation

In our policy network structure, the ILNR graph needs to be created according to its connection relationship. However, the IOBUF connection relationship is not directly provided in the FPGA design file. If we manually analyze the circuit netlist design to find the connection relationship between IOBUFs, it will inevitably require a very expensive time cost. Therefore, it is necessary to design a method for automatically analyzing circuit netlists and creating the ILNR graph. An ILNR graph is composed of vertices and edges connecting vertices, denoted as $G = (V, E)$. For example, if Fig. 5(a) shows the ILNR graph between four IOBUFs, then the vertex set is $V = [v_0, v_1, v_2, v_3]$ and the edge set is $E = [[v_0, v_1], [v_1, v_2], [v_2, v_3]]$. This work introduces PyTorch Geometric (PyG), a geometric deep learning extension library for PyTorch [26], which creates



(a) IOBUF pin connection



(b) IOBUF netlist connection

Fig. 5. A ILNR schematic graph.

GNN and uses the [*Source*, *Target*] method in PyG to create ILNR graphs. The *Source* is the set of source nodes, and the *Target* is the set of target nodes connected to the source nodes. In addition, we consider that the connections between IOBUFs are bidirectional. Therefore, the ILNR graph in Fig. 5(a) can be represented as $[Source, Target] = [[v_0, v_1, v_2, v_1, v_3, v_2], [v_1,\ v_0, v_1, v_2, v_2, v_3]]$.

If only the direct connection relationship between pins of IOBUFs is considered, then the GNN can only capture the connection relationship between IOBUFs and cannot capture the connection relationship between IOBUFs and other instances. To enable the GNN to accurately capture the connection relationship between IOBUFs and other instances, we define the criterion for determining whether there is a connection between two IOBUFs as follows: When at least one of the netlists connected to the pins of the two IOBUFs is the same, it is considered that the two IOBUFs are connected. Because a netlist typically consists of many different instances. Defining the connection relationships between IOBUFs through IOBUF netlist connections enables the agent to indirectly capture the connection relationships between IOBUFs and other instances. Therefore, we design a new ILNR graph creation algorithm based on the IOBUF netlist (**Algorithm 1**).

In Algorithm 1, we use existing placement engines based on GO (e.g., DREAMPlaceFPGA [2]) to obtain the netlist mapping set Ω_i^{net} of each IOBUF pin and its connection. Assuming that the netlists connected to each IOBUF pin in Fig. 5(a) are $\Omega_0^{net} = [net_0, net_1]$, $\Omega_1^{net} = [net_0, net_3]$, $\Omega_2^{net} = [net_1, net_2]$, $\Omega_3^{net} = [net_3, net_4]$. According to Algorithm 1, if $i \neq j$ exists and $\Omega_0^{net} \cap \Omega_j^{net} \neq \emptyset$ when $i = 0$, $j = 0$ to 3 (line 4 of the code), it indicates that there is a connection between the i-th IOBUF and the j-th IOBUF. At this moment, $Source = [0, 1, 0, 2]$ and $Target$=[1, 0, 2, 0]. Similarly, when $i = 3$, we obtain $Source = [0, 1, 0, 2, 1, 0, 1, 3, 2, 0, 3, 1]$ and $Target$=[1, 0, 2, 0, 0, 1, 3, 1, 0, 2, 1, 3], and the corresponding ILNR graph is shown in Fig. 5(b). It is not difficult to find that the above [*Source*, *Target*] has multiple edges, which represent stronger relationships to enhance their weights in the process of graph convolution.

D. Parallelizable Reward Function Design of DRL

We design a policy network structure with multi-action sampling, which requires corresponding reward values for multiple actions at each time step. The traditional DRL only has actual reward in the last time step of completing an episode, and rewards are usually set to zero in other time steps, which can lead to sparse reward problems that are not conducive to agents exploring optimal solutions. The DeepPlace [12] used for ASIC placement generates a non-zero internal reward through RND [20] at each time step to prevent

Algorithm 1 A ILNR Graph Creation Based on the IOBUF Netlist

Input: The netlist mapping set $\Omega_i^{net}(i = 1, 2, \cdots, n_{IOBUF})$ of each IOBUF pin and its connection.

Output: *Source*, *Target*

```
1: $Source$ =[], $Target$=[]
2:   for $i = 0$ to $n_{IOBUF} - 1$ do
3:     for $j = 0$ to $n_{BUF} - 1$ do
4:       if $i \neq j$ and $\Omega_i^{net} \cap \Omega_j^{net} \neq \emptyset$ then
5:         $Source = Source \cup \{i, j\}$
6:         $Target = Target \cup \{j, i\}$
```

sparse reward problems. However, this method is designed to sample only one action at each time step, and performing RND will increase the running time. To reward multiple actions sampled at each time step and avoid sparse reward problems, we design a parallelizable reward function as shown in (8).

$$r_{i,t} = (-1) \cdot \left[W_t(x, y) - 10^6 \right] \cdot \tau, i = 1, 2, \cdots, n_{IOBUF} \tag{8}$$

$$W_t(x, y) = W(x, y) \tag{9}$$

where $r_{i,t}$ is the reward for the i-th IOBUF at time t. τ is the scaling factor and takes the value 10^{-6} in this work. $W_t(x, y)$ is the HPWL of FPGA GP at time t.

According to (8), the larger the GP HPWL value, the smaller the reward value; otherwise, the reward value will be larger. This means that the placement actions of each IOBUF are evaluated by the final GP HPWL. The purpose of designing a parallelizable reward function in this way is to optimize the IOBUF placement with the goal of obtaining the optimal GP solution. Having the shortest HPWL between IOBUFs may not necessarily mean that the final GP HPWL is optimal.

The total IOBUF number in (8) should be an integer divisible by the number of parallelization. As shown in (10), for the total IOBUF number that cannot be integer divided by the numbers of parallelization, the solution is to increase virtual IOBUFs. The reward for the placement action of the virtual IOBUF is zero, and the placement position is fixed to $(x, y) = (0, 0)$. When conducting GP optimization based on GO, the placement positions of virtual IOBUFs will not be written into the FPGA architecture.

$$n_{IOBUF} = \begin{cases} n_{IOBUF}, if\ n_{IOBUF} \% n_{parallel} = 0 \\ \left(n_{IOBUF} // n_{pl} + 1 \right) \cdot n_{pl},\ else \end{cases} \tag{10}$$

However, only by completing the placement of all IOBUFs can we obtain the GP HPWL based on GO. This means that the reward value cannot be immediately obtained according to (8) at each time step. Therefore, we have designed an IOBUF placement action parallel optimization flow, as shown in Fig. 6. In Fig. 6. Firstly, executing the IOBUF placement based on DRL at each time step to obtain a $n_{pl} \times 1$ action. Correspondingly, we temporarily set the reward value for each time step to $\mathbf{0}_{n_{pl} \times 1}$, which is called the old reward. Then, the CDR placement based on GO will be executed to complete the placement of all FPGA instances and obtain the HPWL. We calculate the actual rewards for each IOBUF according to (8), and the old rewards are replaced by actual rewards. Finally, the policy network is updated through PPO.

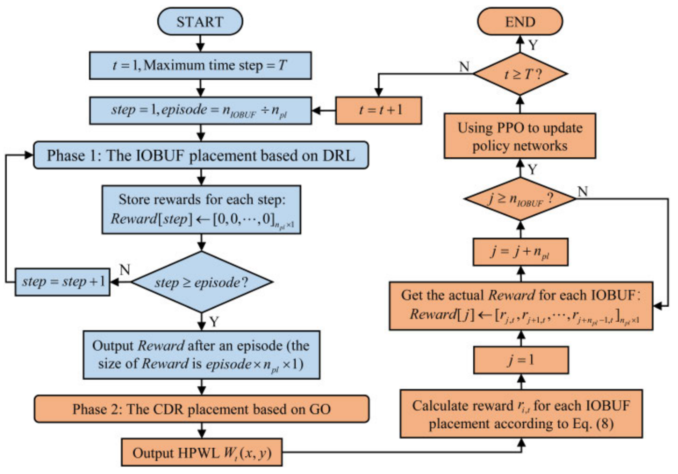

START
$t = 1$, Maximum time step $= T$
$step = 1, episode = n_{IOBUF} \div n_{pl}$
Phase 1: The IOBUF placement based on DRL
Store rewards for each step: $Reward[step] \leftarrow [0, 0, \cdots, 0]_{n_{pl} \times 1}$
$step \geq episode$?
N
$step = step + 1$
Y
Output $Reward$ after an episode (the size of $Reward$ is $episode \times n_{pl} \times 1$)
Phase 2: The CDR placement based on GO
Output HPWL $W_t(x, y)$
Calculate reward $r_{i,t}$ for each IOBUF placement according to Eq. (8)
$j = 1$
Get the actual $Reward$ for each IOBUF: $Reward[j] \leftarrow [r_{j,t}, r_{j+1,t}, \cdots, r_{j+n_{pl}-1,t}]_{n_{pl} \times 1}$
$j = j + n_{pl}$
$j \geq n_{IOBUF}$?
N
Y
Using PPO to update policy networks
$t \geq T$?
N
Y
$t = t + 1$
END

Fig. 6. The IOBUF placement action parallel optimization flow.

In Fig. 6, each IOBUF has an actual reward for evaluating the quality of the placement action before using PPO to update the policy network, and the sparse reward problem is avoided.

E. IOBUF Placement Legalization

The proposed policy network structure with multi-action sampling can improve the running speed of DRL, but it may lead to overlapping IOBUF placement actions. Therefore, we propose an IPL method that can quickly avoid overlapping IOBUF placement positions and make the placement positions meet FPGA architecture constraints. The IPL method mainly includes two legalization constraints:

- IOBUFs need to be placed in the IOBUF region boxes of FPGA architecture, and any two IOBUFs cannot overlap.
- Using three-dimensional coordinates $[x, y, z]$ to represent the position of IOBUF in GP results.

1) IOBUF Placed in IOBUF Region Boxes and no Overlap Between IOBUFs: Due to the design of the policy network structure with multi-action sampling in DRL, agents may choose the same placement action for different IOBUFs, resulting in overlap between IOBUFs. To quickly adjust the IOBUF placement position to avoid overlapping situations during DRL training and testing, we design a method to adjust the placement position by calculating the shortest distance between the mapped IOBUF placement position on the FPGA architecture and the legal placement positions of all IOBUFs on the FPGA architecture (**Algorithm 2**). Assuming that the placement coordinates of the IOBUF placement optimization based on DRL are mapped through the mapping relationship f_x and f_y to obtain the placement coordinates B_{xy} of IOBUF on the FPGA architecture. The legal placement coordinates of all IOBUFs on the FPGA architecture are formed by combining $x \in [66, 103]$ and $y' \in [0, 1, \cdots, 25, 30, \cdots, 475]_{1\times416}$ to form all unique legal placement positions $\Gamma_{832\times2}$ (line 1 of the code). Then calculate the sum of squared distances $d_{|\Gamma|\times1}$ between the i-th placement coordinate of IOBUF and all legal placement positions Γ (line 4 of the code), and calculate the index number idx corresponding to the shortest distance in $d_{|\Gamma|\times1}$ (line 5 of the code). If $d[idx] = 0$, it means that the placement position in Γ is not occupied by other IOBUFs. Therefore, there is no need to adjust the placement coordinates. On the contrary, if $d[idx] \neq 0$ indicates that the placement position in Γ has been occupied. Therefore, we will replace the current placement position of IOBUF with the legal position $\Gamma[idx, :]$ corresponding to the shortest distance (line 7 of the code). Finally, we need to delete the legal positions already occupied by IOBUFs in Γ(line 8 of the code) to ensure that the placement positions of each IOBUF are unique and non-overlapping. Algorithm 2 only uses simple numerical operations, and the IOBUF can be moved only a few times to avoid overlap. This method is simple, effective, and runs fast.

2) Using [x, y, z] to Represent the Position of IOBUF: After completing IOBUF placement optimization based on DRL, we map all placement actions to the FPGA architecture based on the mapping relationships f_x and f_y, and ultimately need to represent the placement position of IOBUF in the form of [x, y, BEL z]. For all placement action $\boldsymbol{X}_{n_{IOBUF}\times 1}$, the coordinate $x_{n_{IOBUF}\times 1}$ that meets the FPGA architecture can be obtained by inverse mapping based on the mapping relationship f_x, while for $\boldsymbol{Y}_{n_{IOBUF}\times 1}$, the coordinate $y'_{n_{IOBUF}\times 1}$ obtained by inverse mapping based on f_y needs to be transformed into $y_{n_{IOBUF}\times 1}$ and BEL$z_{n_{IOBUF}\times 1}$according to (11) and (12).

$$y_{n_{IOBUF}\times 1} = I_{IOBUF}\left(\boldsymbol{Y'}_{n_{IOBUF}\times 1}//I_{IOBUF}\right) \tag{11}$$

$$\mathrm{BEL}z_{n_{IOBUF}\times 1} = \boldsymbol{Y'}_{n_{IOBUF}\times 1}\%I_{IOBUF} \tag{12}$$

where I_{IOBUF} is the interval between two IOBUFs in the site y coordinate direction. In the Xilinx UltraScale architecture, $I_{IOBUF} = 30$.

F. GO for CDR Placement

After completing the IOBUF placement based on DRL, the proposed DrlGoFPGA needs to use the GO method to place the remaining CDR instances in the FPGA. The current GO methods (e.g., *elfplace* [3]) exhibit higher computational efficiency and solution performance compared to traditional heuristic methods (e.g., simulated annealing algorithm). Therefore, we chose the GO method to complete the final GP and used the obtained GP metric HPWL to calculate the reward value. In this work, we obtained the final GP results using various state-of-the-art placers based on GO, including DREAMPlaceFPGA [2], OpenPARF [4], DREAMPlaceFPGA-MP [30], and OpenPARF 3.0 [31].

Algorithm 2 Avoid Overlap between IOBUFs

Input: The IOBUF site coordinate x and y', the placement coordinates B_{xy} of IOBUF, the total IOBUF number n_{IOBUF}
Output: All placement coordinates B_{xy}^{lg} of IOBUF after legalization

```
1: $\Gamma = \{(x, y')|x \in [66, 103], y' \in [0, \cdots, 25, 30, \cdots, 475]\}_{832\times 2}$
2: $B_{xy}^{\lg} \leftarrow B_{xy}$
3: for $i$ =0 to $n_{IOBUF} - 1$ do:
4:     $d_{|\Gamma|\times 1} = \sum_{j=0}^{1}(B_{xy}^{lg}[i, j] - \Gamma[:, j])^2$
5:     $idx = \arg\min d_{|\Gamma|\times 1}$
6:     if $d[idx] \neq 0$ then:
7:         $B_{xy}^{lg}[i, :] = \Gamma[idx, :]$
8:     $\Gamma = \Gamma[0 : idx - 1, :] \cup \Gamma[idx + 1 : end, :]$
```

TABLE I
THE COMPOSITION OF ISPD'2016 BENCHMARKS

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

TABLE II
THE COMPOSITION OF ISPD' 2017 BENCHMARKS

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

IV. EXPERIMENTAL RESULTS

A. Benchmarks and Experimental Setting

We conduct experiments using the ISPD'2016 [27] and ISPD'2017 [28] benchmarks. TABLE I and TABLE II 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 one NVIDIA Corporation GP102 [GeForce RTX 3090] GPU. PyTorch [18] and PPO [24] are used for all experiments. We utilize GCN [25] as the GNN backbone. For CDR instances, we adopt GO such as DREAMPlaceFPGA [2] and OpenPARF [4] for experiments. The IOBUF placement optimization based on PPO uses the Adam optimizer [29]. Due to the varying complexity of each circuit placement problem in the benchmarks, the learning rate setting of the Adam optimizer has a significant effect on the results. We used the adaptive reduction learning rate method **ReduceLROnPlateau** in PyTorch. TABLE III, TABLE IV, and TABLE VI compares the GP and routing results of DrlGoFPGA with DREAMPlaceFPGA and OpenPARF. TABLE V compares the results of DrlGoFPGA with SA/IOBUF-DREAMPlaceFPGA considering IOBUF optimization. TABLE IX and TABLE X compares the results of DrlGoFPGA with DREAMPlaceFPGA-MP and OpenPARF 3.0. In all comparison metrics, 'WL' is the wirelength after routing the placed designs using the router in OpenPARF. 'HPWL' is the half-perimeter wirelength after GP. 'GPT' is GP time.

TABLE III
Experimental Results of DREAMPlaceFPGA and DrlGoFPGA on ISPD’2016 Benchmarks (HPWL and WL in 10^3)

Design	DREAMPlaceFPGA [2]			DrlGoFPGA01-PT			DrlGoFPGA01-FT		
	WL	HPWL	GPT (s)	WL	HPWL	GPT (s)	WL	HPWL	GPT (s)
FPGA01	339.4	190.9	8.7	335.5	188.7	7.2	335.5	188.7	7.2
FPGA02	698.8	490.4	22.8	698.0	491.5	19.7	681.3	482.9	17.8
FPGA03	3101.2	2053.4	21.6	3024.4	1997.0	17.2	3017.5	1997.5	17.4
FPGA04	5999.8	4023.7	23.1	5974.6	3976.3	17.6	5991.7	3996.8	17.5
FPGA05	12430.9	8122.8	26.6	12058.1	7920.1	19.9	12083.7	7928.4	19.5
FPGA06	6105.6	3403.3	24.1	5940.5	3349.4	19.0	6022.5	3329.6	19.1
FPGA07	10344.3	6214.3	24.1	10286.9	6219.7	19.1	10337.6	6224.6	18.5
FPGA08	9593.3	6735.4	25.0	9494.3	6606.1	19.6	9433.6	6586.1	19.2
FPGA09	12882.8	8132.6	26.7	12870.9	8182.6	21.3	12883.7	8187.8	21.0
FPGA10	5945.6	3326.8	25.6	5923.3	3309.3	19.1	5920.1	3304.0	19.8
FPGA11	13262.2	8921.3	25.5	13321.4	9029.3	20.6	13217.0	8942.8	20.3
FPGA12	7408.4	4384.6	29.2	7292.2	4350.3	23.7	7185.8	4290.3	21.2
Ratio	1.014	1.012	1.290	1.003	1.004	1.023	1.000	1.000	1.000

DrlGoFPGA01-PT/FT: During pre-training and fine-tuning model testing, the GO method uses DREAMPlaceFPGA.

TABLE IV
Experimental Results of DREAMPlaceFPGA and DrlGoFPGA on ISPD’2017 Benchmarks (HPWL and WL in 10^3)

Design	DREAMPlaceFPGA [2]			DrlGoFPGA01-PT			DrlGoFPGA01-FT		
	WL	HPWL	GPT (s)	WL	HPWL	GPT (s)	WL	HPWL	GPT (s)
CLK-FPGA01	2352.7	1748.6	19.7	2302.5	1691.2	16.2	2276.9	1678.6	15.1
CLK-FPGA02	2593.6	1715.9	19.5	2598.1	1731.3	15.2	2625.2	1742.8	14.6
CLK-FPGA03	6431.5	4859.1	19.3	6334.9	4548.9	18.5	6334.4	4544.9	16.1
CLK-FPGA04	4417.4	2989.7	19.1	4395.6	2974.7	15.6	4394.8	2972.7	15.1
CLK-FPGA05	5563.0	3940.1	19.5	5419.4	3739.5	17.0	5397.3	3733.2	15.9
CLK-FPGA06	6697.4	4671.4	19.7	6435.0	4505.6	17.1	6432.1	4510.0	16.6
CLK-FPGA07	2711.8	1842.5	19.2	2724.8	1843.0	16.6	2723.3	1843.5	16.0
CLK-FPGA08	2146.2	1585.8	20.9	2146.9	1586.5	18.3	2119.4	1576.9	18.2
CLK-FPGA09	2712.5	1852.3	19.2	2758.3	1859.1	16.7	2760.7	1876.5	15.5
CLK-FPGA10	5005.1	3437.9	19.3	4850.9	3279.3	17.1	4846.4	3274.7	15.7
CLK-FPGA11	4592.2	3078.1	18.5	4475.9	2959.1	16.1	4477.8	2960.3	15.7
CLK-FPGA12	3593.9	2330.1	18.8	3601.2	2349.9	16.6	3602.0	2349.9	15.7
CLK-FPGA13	4643.4	3291.7	18.5	4465.3	3081.4	15.8	4466.1	3081.1	15.5
Ratio	1.015	1.026	1.222	1.001	1.000	1.054	1.000	1.000	1.000

DrlGoFPGA01-PT/FT: During pre-training and fine-tuning model testing, the GO method uses DREAMPlaceFPGA.
DREAMPlaceFPGA and DrlGoFPGA01-PT/FT do not support clock routing constraints on ISPD’2017 benchmarks.

TABLE V
Experimental Results of SA/IOBUF-DREAMPlaceFPGA and DrlGoFPGA (HPWL and WL in 10^3, GPT in seconds)

Design	SA-DREAMPlaceFPGA			IOBUF-DREAMPlaceFPGA			Design	SA-DREAMPlaceFPGA			IOBUF-DREAMPlaceFPGA		
	WL	HPWL	GPT	WL	HPWL	GPT		WL	HPWL	GPT	WL	HPWL	GPT
FPGA01	353.2	206.4	7.6	340.6	195.5	29.4	CLK-FPGA01	2304.6	1723.2	15.9	2277.6	1681.4	104.0
FPGA02	707.8	511.5	20.8	696.8	496.9	85.4	CLK-FPGA02	2632.7	1745.7	15.9	2590.7	1716.0	99.0
FPGA03	3047.7	2016.0	17.2	3055.3	2016.2	136.2	CLK-FPGA03	6275.0	4446.0	16.3	6265.0	4434.2	125.5
FPGA04	6025.9	4015.6	17.4	5996.5	4001.2	131.8	CLK-FPGA04	4364.3	3004.9	15.0	4470.7	3049.1	120.4
FPGA05	12221.0	8011.7	20.2	12176.6	8005.7	151.9	CLK-FPGA05	5357.9	3696.0	16.0	5343.7	3689.9	127.0
FPGA06	5913.2	3325.7	18.8	5964.0	3348.0	189.9	CLK-FPGA06	6442.2	4502.0	16.2	6396.7	4444.8	124.3
FPGA07	10259.8	6232.6	18.2	10303.9	6192.4	188.0	CLK-FPGA07	2727.7	1850.1	16.5	2725.7	1852.5	100.5
FPGA08	9459.2	6600.6	20.8	9518.2	6599.7	143.4	CLK-FPGA08	2157.3	1589.1	17.7	2180.9	1730.5	107.8
FPGA09	12810.2	8139.1	21.2	12753.7	8096.2	189.9	CLK-FPGA09	2745.5	1863.6	16.0	2751.2	1870.0	102.2
FPGA10	5939.0	3324.2	20.2	5858.4	3284.9	193.4	CLK-FPGA10	4823.0	3198.2	15.9	4817.4	3167.4	121.3
FPGA11	13242.6	8977.0	19.8	13167.3	8867.9	201.1	CLK-FPGA11	4492.8	2970.8	16.9	4478.0	2966.4	119.3
FPGA12	7298.8	4247.0	24.2	7175.5	4228.4	142.0	CLK-FPGA12	3574.8	2324.5	15.9	3545.0	2326.9	111.2
Ratio	1.009	1.014	1.038	1.002	1.004	7.921	CLK-FPGA13	4456.0	3088.4	15.8	4428.1	3050.4	123.5
							Ratio	1.000	0.999	1.022	0.998	1.001	7.236

Ratio: Calculated based on the experimental results of DrlGoFPGA01-FT in TABLE III and TABLE IV.
SA/IOBUF-DREAMPlaceFPGA do not support clock routing constraints on ISPD’2017 benchmarks.

B. Pre-Training and Fine-Tuning of IOBUF Placement Model

We used the proposed DrlGoFPGA for pre-training the IOBUF placement model on the FPGA01 design of ISPD’2016 benchmarks (named DrlGoFPGA01-PT), where the GO method used is DREAMPlaceFPGA [2]. We chose to pre-train on the FPGA01 because it has the smallest scale. The initial learning rate for pre-training is set to 7.5×10^{-4}, and the minimum value for the adaptive reduction learning rate is set to 1×10^{-5}. The number of

TABLE VI
EXPERIMENTAL RESULTS OF OPENPARF AND DRLGOFPGA ON ISPD'2016/2017 BENCHMARKS (HPWL AND WL IN 10^3)

Design	OpenPARF [4]			DrlGoFPGA01-PT			Design	OpenPARF [4]			DrlGoFPGA01-PT		
	WL	HPWL	GPT (s)	WL	HPWL	GPT (s)		WL	HPWL	GPT (s)	WL	HPWL	GPT (s)
FPGA01	329.7	213.3	15.7	323.8	208.5	15.7	CLK-FPGA01	2134.2	1602.7	28.6	2170.1	1626.1	29.3
FPGA02	693.4	482.5	25.2	678.2	482.2	22.1	CLK-FPGA02	2565.6	1727.2	29.4	2571.6	1730.4	30.2
FPGA03	3003.0	1978.5	25.5	2958.0	1941.8	26.6	CLK-FPGA03	6115.3	4410.8	40.4	6008.4	4339.2	40.3
FPGA04	6014.6	3986.2	25.1	5957.1	3961.6	26.1	CLK-FPGA04	4356.2	3022.6	42.7	4302.4	2998.4	39.6
FPGA05	12188.6	7951.0	33.2	11827.5	7715.4	30.1	CLK-FPGA05	5293.8	3750.3	36.6	5284.5	3723.0	39.1
FPGA06	5922.4	3313.9	28.6	5853.5	3280.4	30.1	CLK-FPGA06	6378.5	4531.2	40.4	6336.1	4532.9	35.2
FPGA07	10084.7	6159.6	27.9	10050.6	6150.9	29.0	CLK-FPGA07	2660.7	1849.9	28.5	2659.1	1836.3	29.6
FPGA08	9324.9	6554.8	31.8	9265.7	6455.3	32.3	CLK-FPGA08	2035.7	1498.9	26.8	2038.2	1496.6	27.6
FPGA09	12540.6	7979.3	31.8	12563.0	7971.4	32.5	CLK-FPGA09	2604.2	1833.4	29.7	2607.6	1830.2	29.8
FPGA10	5574.6	3212.9	30.0	5541.4	3181.1	30.9	CLK-FPGA10	4664.1	3214.1	33.8	4641.7	3194.7	33.3
FPGA11	13145.4	8923.4	32.6	12938.3	8846.7	33.7	CLK-FPGA11	4365.9	3037.4	32.6	4317.0	3010.3	32.3
FPGA12	6961.2	4248.9	33.8	6802.4	4164.8	34.1	CLK-FPGA12	3475.3	2365.1	30.2	3498.7	2388.8	31.5
Ratio	1.013	1.012	0.998	1.000	1.000	1.000	CLK-FPGA13	4440.9	3103.0	32.8	4421.9	3100.1	35.5
							Ratio	1.002	1.002	0.997	1.000	1.000	1.000

DrlGoFPGA01-PT: During pre-training model testing, the GO method uses OpenPARF.
OpenPARF and DrlGoFPGA01-PT both support clock routing constraints on ISPD'2017 benchmarks.

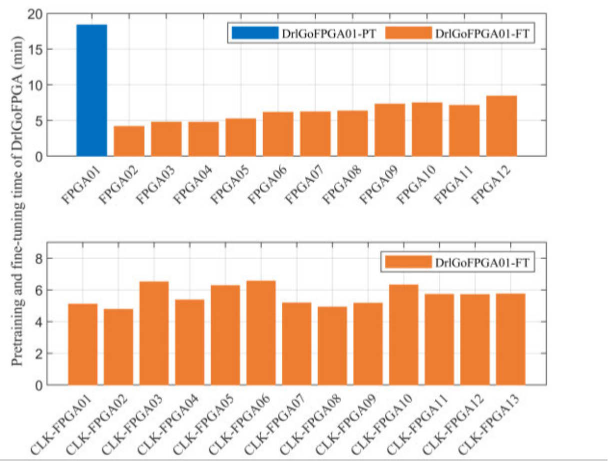


Fig. 7. The pre-training and fine-tuning time of DrlGoFPGA.

parallelization is $n_{pl} = 12$. The dimension of embedding ILNR knowledge is $n_{ILNR} = 12$. The total number of episodes is set to 100. To verify the generalization of the DrlGoFPGA01-PT, we designed the following method: First, the DrlGoFPGA01-PT was retrained on different circuits to obtain the fine-tuned models (collectively named DrlGoFPGA01-FT). Then, the DrlGoFPGA01-FT is applied to the corresponding circuits for GP optimization, and compared with the results obtained directly using DrlGoFPGA01-PT on different circuits. To prevent performance degradation caused by significant parameter changes, we set the initial learning rate for fine-tuning to 7.5×10^{-5} and the number of fine-tuning episodes to 10. Fig. 7 shows that DrlGoFPGA01-PT completes training in about 18 minutes, and DrlGoFPGA01-FT completes fine-tuning in 2.4 hours.

C. Comparison With DREAMPlaceFPGA

TABLE III and TABLE IV compare the experimental results of DREAMPlaceFPGA and DrlGoFPGA. All placers use *elfplace*-CPU [3] for LG and DP. To ensure the successful routing of all circuits, the routability optimization in the GO method was enabled during the GP optimization phase.

In TABLE III and TABLE IV, compared to DREAMPlaceFPGA, the GPT of DrlGoFPGA01-PT and DrlGoFPGA01-FT decreased by 26.7% and 29.0% respectively on the ISPD'2016 benchmarks, and decreased by 16.8% and 22.2% respectively on the ISPD'2017 benchmarks. The HPWL of DrlGoFPGA01-PT and DrlGoFPGA01-FT decreased by 0.8% and 1.2% respectively on the ISPD'2016 benchmarks, and both decreased by 2.6% on the ISPD'2017 benchmarks. The WL of DrlGoFPGA01-PT and DrlGoFPGA01-FT decreased by 1.1% and 1.4% respectively on the ISPD'2016 benchmarks, and decreased by 1.4% and 1.5% respectively on the ISPD'2017 benchmarks.

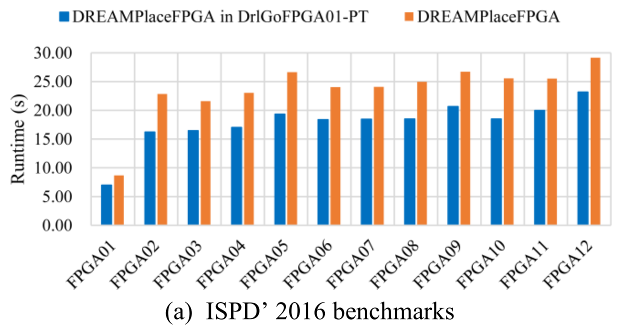
The comparison of runtime between DREAMPlaceFPGA in DrlGoFPGA01-PT and DREAMPlaceFPGA is shown in Fig. 8. When optimizing the same circuit, DREAMPlaceFPGA in DrlGoFPGA requires less runtime. Therefore, a more optimal IOBUF position is beneficial for improving the speed of GP optimization and the quality of placement and routing.

In TABLE III and TABLE IV, the differences between DrlGoFPGA01-PT and DrlGoFPGA01-FT on HPWL and WL are only 0%-0.4% and 0.1%-0.3%, respectively. This proves that DrlGoFPGA01-PT has good generalization performance.

D. Comparison With SA/IOBUF-DREAMPlaceFPGA

Currently, there is a lack of research on the effect of IOBUF placement on GP. To further explore whether the proposed method for IOBUF placement optimization has advantages, we conducted the following two comparative experiments:

1) SA-DREAMPlaceFPGA: We first uses the simulated annealing (SA) algorithm to optimize the placement of the IOBUF. The proposed IPL method (see Part E of Section III) was used to legalize the placement position of IOBUF. Once the IOBUF placement optimization is completed, the IOBUF position will be fixed, and then DREAMPlaceFPGA will be used to complete the GP optimization. Finally, we will use the GP metric HPWL as the objective function feedback to SA for further iterative optimization. The parameter settings for SA are as follows: The initial temperature is $T_0 = 100$ and the



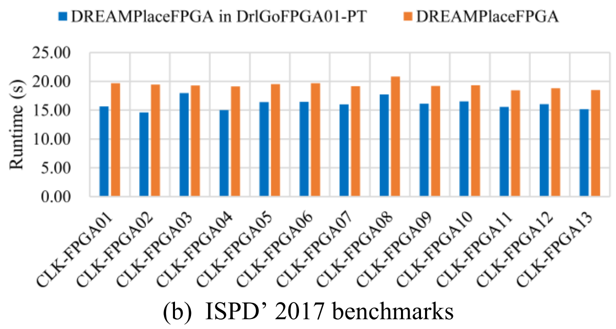


Fig. 8. The comparison of optimization time between DREAMPlaceFPGA in DrlGoFPGA01-PT and DREAMPlaceFPGA.

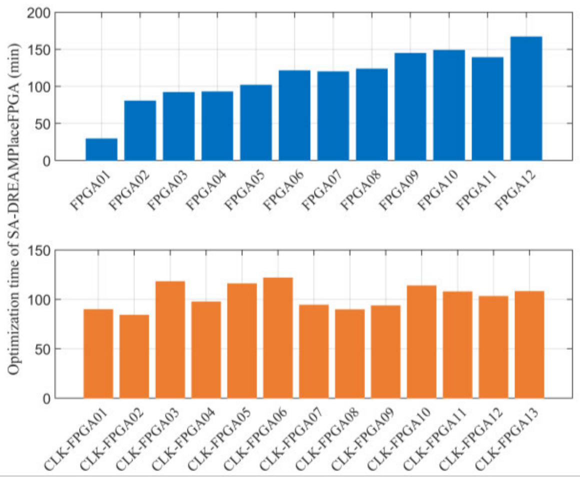


Fig. 9. The optimization time distribution diagram of FPGA placement optimization based on SA-DREAMPlaceFPGA.

termination temperature is $T_f = 0.1$. Due to DrlGoFPGA01-PT needing to be trained 100 times, SA-DREAMPlaceFPGA needs to perform 100 optimization iterations. Because of $T_0\alpha^{99} \approx 0.104 > T_f$ and $T_0\alpha^{100} \approx 0.097 < T_f$, the cooling coefficient is set to $\alpha = 0.933$.

2) *IOBUF-DREAMPlaceFPGA:* We use DREAMPlace-FPGA to optimize IOBUF and CDR instances together. Specifically, we have added each IOBUF placement position variable $[x, y' = y + z]$ to the optimization variable of the CDR instance placement position in the DREAMPlaceFPGA. Then, the Adam optimizer and *elfplace* method in DREAM-PlaceFPGA will be used to perform GO and update the IOBUF variables. Finally, after each update of the IOBUF variable, the proposed IPL method (see Part E of Section III) is used to legalize the placement position of IOBUF (first, Algorithm 2 is used to quickly move IOBUF to avoid overlap. Then, based on (11) and (12), the optimization variable $[x, y']$ of IOBUF is represented back to $[x, y, \text{BEL } z]$).

We respectively used the *elfplace*-CPU [3] and router of OpenPARF [4] to perform LG, DP, and routing. TABLE V compares the experimental results of SA/IOBUF-DREAMPlaceFPGA (the results of SA-DREAMPlaceFPGA obtained by testing the GP results after optimization) and DrlGoFPGA. Fig. 9 shows the optimization time distribution of GP optimization based on SA-DREAMPlaceFPGA.

In TABLE V, compared to SA-DREAMPlaceFPGA, the GPT of DrlGoFPGA01-FT decreased by 3.8% on the ISPD'2016 and 2.2% on the ISPD'2017 benchmarks. The HPWL of DrlGoFPGA01-FT decreased by 1.4% on the ISPD'2016 and was almost the same on the ISPD'2017 benchmarks. The WL of DrlGoFPGA01-FT decreased by 0.9% on the ISPD'2016 and was the same on the ISPD'2017 benchmarks. Compared to IOBUF-DREAMPlaceFPGA, DrlGoFPGA01-FT achieved over 7x acceleration on the ISPD'2016 and ISPD'2017 benchmarks, respectively. The HPWL of DrlGoFPGA01-FT decreased by 0.4% on the ISPD'2016 and 0.1% on the ISPD'2017 benchmarks. The WL of DrlGoFPGA01-FT decreased by 0.2% on the ISPD'2016 and increased by 0.2% on the ISPD'2017 benchmarks.

Although IOBUF-DREAMPlaceFPGA can achieve comparable results to DrlGoFPGA01-FT on HPWL and WL, the GPT is relatively long. The results of SA-DREAMPlaceFPGA achieved comparable results to DrlGoFPGA01-FT on the ISPD'2017 benchmarks. However, in Fig. 9, the total optimization time of SA-DREAMPlaceFPGA on the ISPD'2016 and ISPD'2017 benchmarks are about 45 hours, which is about 42 hours longer than the pre-training and fine-tuning time of DrlGoFPGA. The inability of SA-DREAMPlaceFPGA to obtain end-to-end IOBUF placement models will result in increasingly expensive optimization time costs.

In summary, the proposed DrlGoFPGA performs better compared to SA/IOBUF-DREAMPlaceFPGA.

E. Comparison With OpenPARF

To further demonstrate the performance of DrlGoFPGA, were placed the GO method with OpenPARF during model testing, where OpenPARF has integrated LG, DP, and routing tools. TABLE VI presents the relevant experimental results.

As shown in TABLE VI, compared to OpenPARF, the GPT of DrlGoFPGA01-PT increased by 0.2% and 0.3% respectively on the ISPD'2016 and ISPD'2017 benchmarks. The HPWL of DrlGoFPGA01-PT decreased by 1.2% and 0.2% respectively on the ISPD'2016 and ISPD'2017 benchmarks. The WL of DrlGoFPGA01-PT decreased by 1.3% and 0.2% respectively on the ISPD'2016 and ISPD'2017 benchmarks. In TABLE VII, 'CIIs' are clock illegal instances. 'MVD' is the maximum violation distance in CIIs. 'AVD' is the average violation distance in CIIs. Compared to OpenPARF, DrlGoFPGA01-PT can reduce 3 CIIs in the GP phase of the ISPD'2017 benchmarks. Although DrlGoFPGA01-PT increased MVD by 3.4 μm, it decreased AVD by 0.1 μm.

Therefore, DrlGoFPGA01-FT exhibits better performance in different GO methods and has an advantage in handling clock routing constraints during the GP phase.

TABLE VII

CLOCK ROUTING CONSTRAINT ANALYSIS[1] OF OPENPARF AND DRLGOFPGA IN THE GLOBAL PLACEMENT PHASE

ISPD'2017 benchmarks	OpenPARF [4]			DrlGoFPGA01-PT		
	CIIs	MVD	AVD	CIIs	MVD	AVDA
Avg.	23	37.9	9.1	20	41.3	9.0

DrlGoFPGA01-PT: During model testing, the GO method uses OpenPARF.

TABLE VIII

THE MLCAD 2023 PUBLIC BENCHMARK SUITE STATISTICS

Design	Statistics
Number	141
#Instances	558K-711K
LUT/FF/BRAM/DSP (%)	70%-84%/38%-47%/80%-90%/80%-90%
Rent^5 /#Regions	0.65-0.72/0-22
#Instances within Regions	0-285K
Instances within Regions (%)	0%-44.29%
Cascaded DSP Macros Size	{2, 5, 7, 10, 60} ×
Cascaded BRAM Macros Size	{2, 5, 7, 10, 30} ×

F. Comparison With DREAMPlaceFPGA-MP/OpenPARF 3.0

We extend DrlGoFPGA to DREAMPlaceFPGA-MP (MP) and OpenPARF 3.0 (OP 3.0), respectively. The MP is an open-source GPU-accelerated placer supporting cascade-shaped macros and region constraints, including IOBUFs. The OP 3.0 employs GO with look-ahead legalization to handle IOBUFs, achieving competitive results with minimal runtime overhead. The MP and OP 3.0 were developed for the MLCAD 2023 benchmark [32]. The target FPGA for the benchmark is the Ultrascale+ xcvu3p ffvc1517-1-i device. The placement coordinate distribution of the IOBUF is $[x \in [68,\ 138], y \in [0, 30, \cdots, 270]_{1\times 10}, \text{BEL}z \in [0, 1, \cdots, 25]_{1\times 26}]$, and it is converted to $[x, y' = y + z]$. According to DrlGoFPGA, establish coordinate mapping sets between the IOBUF canvas and the actual architecture: $f_x : [0, 1] \rightarrow [68, 138]$ and $f_y : [0, 1, \cdots,\ 260]_{1\times 260} \rightarrow [0, 1, \cdots, 25, 30, \cdots, 275]_{1\times 260}$, and define the IOBUF canvas size in DRL as $n_{pl} \times 1 \times 2 \times 260$. The MLCAD 2023 public benchmark originally consisted of 180 public designs. Due to MP and OP 3.0 not currently supporting the placement of URAM instance types, we referred to OP 3.0 for benchmark corrections. At present, there are 141 designs that can be implemented on MP and OP 3.0, respectively, and the benchmark composition is shown in TABLE VIII. We run the DrlGoFPGA ($n_{pl} = n_{ILNR} = 12$)on Design_1, where the GO method used during training is MP, and the pre-trained model is named DrlGo-Design_1-PT. TABLE IX and TABLE X respectively provide a comparison of DrlGoFPGA with MP and OP 3.0. Due to the license restriction of Vivado in MLCAD 2023 contests, we cannot get the routed WL from Vivado and use HPWL to approximate the final routed WL.

TABLE IX shows that compared to MP, using the pre-trained model Drlgo-Design_1-PT for other design GP optimizations achieved a 13.2% reduction in GPT and a 1.4% reduction in HPWL. TABLE X shows that compared to OP

[1]The clock routing constraint analysis for each design on ISPD'2017 benchmarks, as well as the HPWL and GPT value for each design on MLCAD 2023 benchmarks are available at: https://dx.doi.org/10.21227/049d-r758

TABLE IX

EXPERIMENTAL RESULTS[1] OF DREAMPLACEFPGA-MP AND DRLGOFPGA ON MLCAD 2023 BENCHMARKS

Metrics	DREAMPlaceFPGA-MP [30]		DrlGo-Design_1-PT	
	Geo. Mean	Ratio	Geo. Mean	Ratio
GPT (s)	39.256	1.132	34.820	1.000
HPWL (in 10^3)	5486.139	1.014	5414.813	1.000

DrlGo-Design_1-PT: During model testing, the GO method uses MP.

TABLE X

EXPERIMENTAL RESULTS[1] OF OPENPARF 3.0 AND DRLGOFPGA ON MLCAD 2023 BENCHMARKS

Metrics	OpenPARF 3.0 [31]		DrlGo-Design_1-PT	
	Geo. Mean	Ratio	Geo. Mean	Ratio
GPT (s)	39.628	0.989	40.091	1.000
HPWL (in 10^3)	6119.375	1.021	5998.222	1.000

DrlGo-Design_1-PT: During model testing, the GO method uses OP 3.0.

3.0, DrlGo-Design_1-PT achieved a 2.1% reduction in HPWL and a 1.1% increase in GPT. Therefore, DrlGoFPGA has good scalability on different FPGA architectures, and can still achieve better GP results compared to MP and OP 3.0.

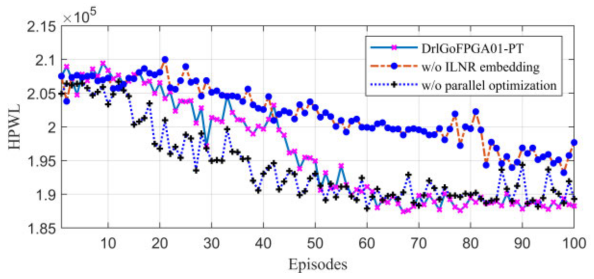
G. Ablation Study

To explore the effect of various factors on the DrlGoFPGA, we conducted a series of ablation studies on reward function, embedding ILNR knowledge dimensions, IOBUF legalization constraints, and the combination of different NNs and GNNs.

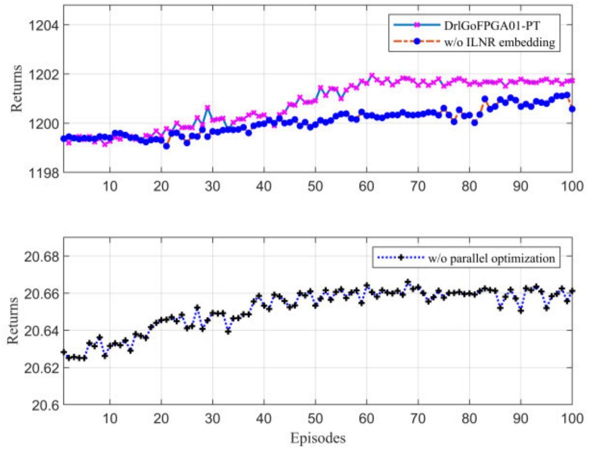
1) The Effect of Parallel Optimization and ILNR Embedding on Solution Performance: Fig. 10 shows the comparison of the IOBUF placement model training results on the FPGA01 between the DrlGoFPGA01-PT, the DrlGoFPGA01-PT without ILNR embedding (denoted as w/o ILNR embedding), and the DrlGoFPGA01-PT without parallel optimization (denoted as w/o parallel optimization). Without the embedding of ILNR information, it is difficult for the agent to explore optimal solutions within 100 episodes. Compared to w/o parallel optimization, DrlGoFPGA01-PT not only explores better solutions but also has higher stability during algorithm optimization. Therefore, the parallelizable reward function and embedding ILNR knowledge into CNN proposed in this paper can improve the ability of agents to explore optimal solutions.

2) The Effect of the Policy Network Structure With Multi-Action Sampling on Solving Speed of DRL: We studied the effect of the numbers of parallelization on the training, fine-tuning, or testing time of DRL (this ablation study is named DrlGoFPGA-n_{pl}). TABLE XI shows the experimental results of DrlGoFPGA-n_{pl} on the FPGA01, FPGA03, and FPGA06, where $n_{ILNR} = 12$ and $n_{pl} = 1, 2, 4, 8, 12, 16, 32$. As the number of parallelization increases, the time required to complete an IOBUF placement based on DRL in one time step gradually decreases. Therefore, the policy network structure with multi-action sampling proposed in this paper can accelerate the training, fine-tuning, or testing time of DRL.

3) The Effect of Numerical Settings for the Number of Parallelization and Embedding ILNR Knowledge Dimension on Solution Performance: We designed the following three ablation studies and ultimately provided recommended values:



(a) HPWL



(b) Return

Fig. 10. Comparison of DrlGoFPGA01-PT, w/o ILNR embedding and w/o parallel optimization.

TABLE XI
EXPERIMENTAL RESULTS OF DRLGOFPGA- n_{pl}

n_{ILNR}	n_{pl}	FPGA01 ($n_{IOBUF}=151$)	FPGA03 ($n_{IOBUF}=401$)	FPGA06 ($n_{IOBUF}=601$)
		IPT (s)		
12	1	0.80	1.33	1.83
	2	0.63	0.89	1.17
	4	0.55	0.69	0.86
	8	0.51	0.62	0.69
	12	0.49	0.55	0.63
	16	0.48	0.53	0.59
	32	0.47	0.50	0.56

IPT: The time required to complete an IOBUF placement based on DRL in one time step during model training, fine-tuning or testing.

- Study the HPWL value changes based on DrlGoFPGA on the FPGA01 design when $n_{ILNR} = 12$, $n_{pl} = 2, 4, \cdots, 32$.
- Study the HPWL value changes based on DrlGoFPGA on the FPGA01 design when $n_{pl} = 12$, $n_{ILNR} = 2, 4, \cdots, 32$.
- Study the HPWL value changes based on DrlGoFPGA on the FPGA01 design when $n_{pl} = n_{ILNR} = 2, 4, \cdots, 32$.

We use DrlGoFPGA-n_{pl}-n_{ILNR} to uniformly represent the three ablation experiments. TABLE XII show that the numerical changes in the number of parallelization and embedding ILNR knowledge dimensions have a certain effect on the optimal HPWL obtained by the IOBUF placement model. When $n_{pl} = n_{ILNR} = 12$, HPWL obtains the optimal values. Therefore, we suggest that both n_{ILNR} and n_{pl} take a value of 12.

TABLE XII
EXPERIMENTAL RESULTS OF DRLGOFPGA-$n_{pl} - n_{ILNR}$

(n_{ILNR}, n_{pl})	(12,2)	(12,4)	(12,8)	(12,12)	(12,16)	(12,32)
HPWL	190.7	193.8	190.1	188.7	191.4	191.6
(n_{ILNR}, n_{pl})	(2,12)	(4,12)	(8,12)	(12,12)	(16,12)	(32,12)
HPWL	196.4	189.0	191.2	188.7	189.8	192.8
(n_{ILNR}, n_{pl})	(2,2)	(4,4)	(8,8)	(12,12)	(16,16)	(32,32)
HPWL	193.3	200.3	189.4	188.7	189.7	189.4

HPWL (in 10^3): Obtained by testing with the pre-trained placement model.

4) The Effect of the Proposed IPL Method on Solution Performance: Fig. 11 shows the results of DrlGoFPGA01-PT using the IPL method and using action resampling for legalization on FPGA01. The using action resampling for legalization here refers to the fact that the IOBUF placement based on DRL method does not use the IPL method. Instead, after performing multi-action sampling at each time step, the already sampled actions are masked according to **Algorithm 3** (lines 2-3 and 9-10 of code). If the i-th action overlaps with the already sampled action, it needs to be resampled (lines 5-8 of code). In Fig. 11, using action resampling for legalization is difficult to explore an optimal solution within 100 episodes. Therefore, the proposed IPL method is a rule-embedding that is beneficial for agents to explore optimal solutions.

Algorithm 3 Using Action Resampling for Legalization

Input: The probability distribution $\boldsymbol{probs}_{n_{pi}\times 832}$ of actions at each time step, the action set $\boldsymbol{Action}_{n_{pl}\times 1}$ obtained after performing multi-action sampling, the initialize action mask $mask = \{(x, y)|x = y = 1\}_{832\times 1}$
Output: The legal actions set $\boldsymbol{Action}_{n_{pl}\times 1}$ at each time step

```
1: for $i = 0$ to $n_{pl} - 1$ do:
2:     if $mask[\boldsymbol{Action}[i]] == 1$ then:
3:         $mask[\boldsymbol{Action}[i]] \leftarrow 0$
4:     else:
5:         $flag \leftarrow True$
6:         while $flag$ then:
7:             $\boldsymbol{probs} \leftarrow \boldsymbol{probs} \circ mask$ ▷∘ is Hadamard multiplication
8:             $\boldsymbol{Action}_{n_{pl}\times 1}[i] \leftarrow \boldsymbol{probs}.resample()[i]$
9:             if $mask[\boldsymbol{Action}[i]] == 1$ then:
10:                $mask[\boldsymbol{Action}[i]] \leftarrow 0$, $flag \leftarrow False$
```

Furthermore, we also investigated the effects of different reward function designs and the combination of different NNs (including Transformer [33] and recurrent neural network [34]) and GNN on solution performance, and experimental results[2] show that our work is superior.

Therefore, the above ablation studies validated the effectiveness and superiority of the policy network structure with multi-action sampling, parallelizable reward function, ILNR graph creation method, and IPL method designed in the DrlGoFPGA framework.

[2]The experimental results of the effect of different reward function designs and combinations of different NNs and GNN on solution performance are available at: https://dx.doi.org/10.21227/049d-r758

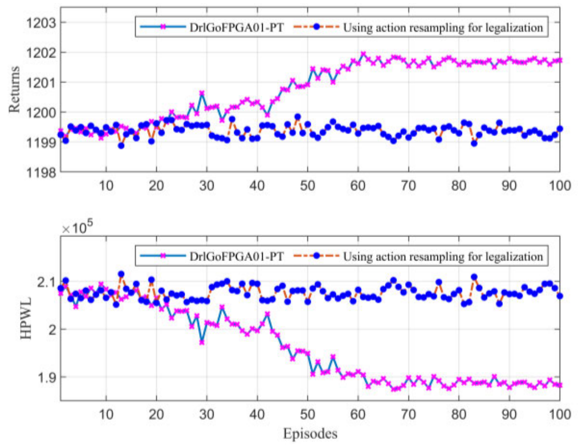


Fig. 11. Comparison of DrlGoFPGA01-PT using the IPL method and using action resampling for legalization.

V. Conclusion

This paper proposes a FPGA GP placement method based on DRL and GO, DrlGoFPGA. In DrlGoFPGA, we designed a policy network structure with multi-action sampling, a parallelizable reward function, an ILNR graph creation method based on the IOBUF netlist, and an IPL method. The experimental results on ISPD'2016/2017 and MLCAD 2023 benchmarks show that compared with DREAMPlaceFPGA, DREAMPlaceFPGA-MP, OpenPARF, and OpenPARF 3.0, DrlGoFPGA reduced GPT by 13.2%-29.0%, HPWL by 0.2%-2.6%, and WL by 0.2%-1.5%. Compared with SA/IOBUF-DREAMPlaceFPGA considering IOBUF placement optimization, DrlGoFPGA improves GP speed by 2.2%-7x, reduces HPWL by 0.1%-1.4%, and WL by 0.2%-0.9%. The SA-DREAMPlaceFPGA requires 45 hours to complete placement optimization for all circuits, which is 42 hours longer than the pre-training and fine-tuning time of DrlGoFPGA. The results of the pre-trained and fine-tuned model have only 0%-0.4% and 0.1%-0.3% differences on HPWL and WL, respectively, proving that the pre-trained model has good generalization performance. We conducted a series of ablation studies on the proposed policy network structure, parallelizable reward function, ILNR graph creation method, and IPL method, verifying the effectiveness and superiority of the proposed DrlGoFPGA framework.

We believe that DrlGoFPGA can provide new directions for developing FPGA physical design engines. In the future, we will focus on researching how to enable pre-trained models to quickly adapt to new circuits and improve generalization.

Acknowledgment

The authors would like to thank the Editor-in-Chief, the associate editor, the senior area editor, and the three anonymous reviewers for their valuable comments and suggestions that greatly contribute to improving the quality of the manuscript.

References

[1] P. Babu and E. Parthasarathy, "Reconfigurable FPGA architectures: A survey and applications," *J. Inst. Eng. B*, vol. 102, no. 1, pp. 143–156, Feb. 2021.

[2] 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. 27th Asia South Pacific Design Autom. Conf. (ASP-DAC)*, Jan. 2022, pp. 300–306.

[3] W. Li, Y. Lin, and D. Z. Pan, "ElfPlace: Electrostatics-based placement for large-scale heterogeneous FPGAs," in *Proc. IEEE/ACM Int. Conf. Computer-Aided Design (ICCAD)*, Nov. 2019, pp. 1–8.

[4] 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," 2023, *arXiv:2306.16665*.

[5] S. Dhar, L. Singhal, M. Iyer, and D. Pan, "FPGA accelerated FPGA placement," in *Proc. 29th Int. Conf. Field Program. Log. Appl. (FPL)*, Sep. 2019, pp. 404–410.

[6] S. Dhar, L. Singhal, M. A. Iyer, and D. Z. Pan, "FPGA-accelerated spreading for global placement," in *Proc. IEEE High Perform. Extreme Comput. Conf. (HPEC)*, Sep. 2019, pp. 1–7.

[7] M. A. Elgammal, K. E. Murray, and V. Betz, "RLPlace: Using reinforcement learning and smart perturbations to optimize FPGA placement," *IEEE Trans. Comput.-Aided Design Integr. Circuits Syst.*, vol. 41, no. 8, pp. 2532–2545, Aug. 2022.

[8] K. E. Murray and V. Betz, "Adaptive FPGA placement optimization via reinforcement learning," in *Proc. ACM/IEEE 1st Workshop Mach. Learn. CAD (MLCAD)*, Sep. 2019, pp. 1–6.

[9] H. Wang et al., "CNN-inspired analytical global placement for large-scale heterogeneous FPGAs," in *Proc. 59th ACM/IEEE Design Autom. Conf.*, Jul. 2022, pp. 637–642.

[10] A. Al-Hyari, A. Shamli, T. Martin, S. Areibi, and G. Grewal, "An adaptive analytic FPGA placement framework based on deep-learning," in *Proc. ACM/IEEE 2nd Workshop Mach. Learn. CAD (MLCAD)*, Nov. 2020, pp. 3–8.

[11] A. Mirhoseini et al., "Chip placement with deep reinforcement learning," 2020, *arXiv:2004.10746*.

[12] R. Cheng and J. Yan, "On joint learning for solving placement and routing in chip design," in *Proc. Adv. Neural Inf. Process. Syst.*, 2021, pp. 16508–16519.

[13] Y. Lai, Y. Mu, and P. Luo, "MaskPlace: Fast chip placement via reinforced visual representation learning," in *Proc. Adv. Neural Inf. Process. Syst.*, 2022, pp. 24019–24030.

[14] Y. Xiao, S. Nazarian, and P. Bogdan, "Self-optimizing and self-programming computing systems: A combined compiler, complex networks, and machine learning approach," *IEEE Trans. Very Large Scale Integr. (VLSI) Syst.*, vol. 27, no. 6, pp. 1416–1427, Jun. 2019.

[15] Y. Xiao, S. Nazarian, and P. Bogdan, "GAHLS: An optimized graph analytics based high level synthesis framework," *Sci. Rep.*, vol. 13, no. 1, p. 2023, Dec. 2023.

[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] M.-K. Hsu, Y.-W. Chang, and V. Balabanov, "TSV-aware analytical placement for 3D IC designs," in *Proc. 48th ACM/EDAC/IEEE Design Autom. Conf. (DAC)*, Jun. 2011, pp. 664–669.

[18] A. Paszke et al., "PyTorch: An imperative style, high-performance deep learning library," in *Proc. Adv. Neural Inf. Process. Syst.*, 2019, pp. 1–12.

[19] Y. Lin, S. Dhar, W. Li, H. Ren, B. Khailany, and D. Z. Pan, "DREAMPlace: Deep learning toolkit-enabled GPU acceleration for modern VLSI placement," in *Proc. 56th Annu. Design Autom. Conf.*, Jun. 2019, pp. 1–6.

[20] Y. Burda, H. Edwards, A. Storkey, and O. Klimov, "Exploration by random network distillation," 2018, *arXiv:1810.12894*.

[21] Y. Xiao, S. Nazarian, and P. Bogdan, "Plasticity-on-Chip design: Exploiting self-similarity for data communications," *IEEE Trans. Comput.*, vol. 70, no. 6, pp. 950–962, Jun. 2021.

[22] Y. Xiao et al., "End-to-end programmable computing systems," *Commun. Eng.*, vol. 2, no. 1, p. 2023, Nov. 2023.

[23] A. Bagheri and N. Masoumi, "Reducing expected delay and power in FPGAs using buffer insertion in single-driver wires," *Microelectron. J.*, vol. 43, no. 12, pp. 1038–1045, Dec. 2012.

[24] J. Schulman, F. Wolski, P. Dhariwal, A. Radford, and O. Klimov, "Proximal policy optimization algorithms," 2017, *arXiv:1707.06347*.

[25] T. N. Kipf and M. Welling, "Semi-supervised classification with graph convolutional networks," 2016, *arXiv:1609.02907*.

[26] A. Paszke et al., "Automatic differentia-tion in Pytorch," in *Proc. 31st Conf. Neural Inf. Process. Syst. (NIPS)*, Long Beach, CA, USA, 2017.

[27] S. Yang, A. Gayasen, C. Mulpuri, S. Reddy, and R. Aggarwal, "Routability-driven FPGA placement contest," in *Proc. Int. Symp. Phys. Design*, Apr. 2016, pp. 139–143.

[28] S. Yang et al., "Clock-aware FPGA placement contest," in *Proc. ACM Int. Symp. Phys. Design*, Mar. 2017, pp. 159–164.

[29] D. P. Kingma and J. Ba, "Adam: A method for stochastic optimization," 2014, *arXiv:1412.6980*.

[30] Z. Xiong, R. Selina Rajarathnam, Z. Jiang, H. Zhu, and D. Z. Pan, "DREAMPlaceFPGA-MP: An open-source GPU-accelerated macro placer for modern FPGAs with cascade shapes and region constraints," 2023, *arXiv:2311.08582*.

[31] J. Mai et al., "OpenPARF 3.0: Robust multi-electrostatics based FPGA macro placement considering cascaded macros groups and fence regions," in *Proc. 2nd Int. Symp. Electron. Design Autom. (ISEDA)*, May 2024, pp. 374–379.

[32] I. Bustany et al., "The 2023 MLCAD FPGA macro placement benchmark design suite and contest results," in *Proc. ACM/IEEE 5th Workshop Mach. Learn. CAD (MLCAD)*, Sep. 2023, pp. 1–6.

[33] A. Vaswani et al., "Attention is all you need," in *Proc. Adv. Neural Inf. Process. Syst.*, 2017, pp. 1–21.

[34] H. Sak, A. Senior, and F. Beaufays, "Long short-term memory based recurrent neural network architectures for large vocabulary speech recognition," 2014, *arXiv:1402.1128*.

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

Jianwang Zhai 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. He is currently an Assistant Professor with the School of Integrated Circuits, Beijing University of Posts and Telecommunications, Beijing. His research interests include architecture modeling, design space exploration, and physical design. He received Best Paper Award Nomination from ASP-DAC 2023 and William J. McCalla Best Paper Award from ICCAD 2021.

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

Zixi Huang received the B.S. degree in computer science and technology from Beijing University of Posts and Telecommunications, Beijing, China, in 2023, where he is currently pursuing the M.S. degree with the School of Artificial Intelligence. His research focuses on large language models (LLM), with interests spanning natural language processing and their applications.

Dan Wu received the B.S. degree in physics from China University of Petroleum, Qingdao, China, in 2019. He is currently pursuing the Ph.D. degree with the School of Artificial Intelligence, Beijing University of Posts and Telecommunications. His research interests include AI for EDA and deepfake detection.

Yida Wang received the B.S. degree in intelligence science and technology from the University of Science and Technology Beijing, Beijing, China, in 2022. He is currently pursuing the M.S. degree with the School of Artificial Intelligence, Beijing University of Posts and Telecommunications. His research interests include reinforcement learning and LLM agents.

Kang Zhao received the Ph.D. degree from the Department of Computer Science and Technology, Tsinghua University, China, in 2009. Then, he was with Tsinghua University, Intel, Xilinx, and AMD, respectively. When working in Xilinx and AMD, he played as a Senior Staff Manager and the Leader of the Vitis HLS Product Team. He is currently a Professor with 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.

Ming Lei received the M.D. degree from the State Key Laboratory of Materials Composites and Advanced Technology, Wuhan University of Technology, in 2004, and the Ph.D. degree from the Laboratory of Nanophysics and Devices, Institute of Physics, Chinese Academy of Sciences, in 2007. He was a Post-Doctoral Fellow with The Hong Kong University of Science and Technology (2007–2008) and The Chinese University of Hong Kong (2009–2010). He is currently a Professor and a Doctoral Supervisor with 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 the Ph.D. degree in pattern recognition and intelligent system from the Institute of Automation, Chinese Academy of Sciences, in 2010. He is currently a Professor with Beijing University of Posts and Telecommunications and the Founder of the Laboratory of Visual Computing and Intelligent System (VCIS). He has authored several top and international conference and journal articles of IEEE TRANSACTIONS ON PATTERN ANALYSIS AND MACHINE INTELLIGENCE and CVPR. His research interests include biometrics, computer vison, intelligent systems, and EDA.