The Policy-gradient Placement and Generative Routing Neural Networks for Chip Design

Anonymous Author(s)
Affiliation
Address
email

Abstract

Placement and routing are two critical yet time-consuming steps of chip design in modern VLSI systems. Distinct from traditional heuristic solvers, this paper on one hand proposes an RL-based model for mixed-size macro placement, which differs from existing learning-based placers that often consider the macro by coarse grid-based mask. While the standard cells are placed via gradient-based GPU acceleration. On the other hand, a one-shot conditional generative routing model, which is composed of a special-designed input-size-adapting generator and a bi-discriminator, is devised to perform one-shot routing to the pins within each net, and the order of nets to route is adaptively learned. Combining these techniques, we develop a flexible neural pipeline, which to our best knowledge, is the first joint placement and routing network without involving any traditional heuristic solver. Experimental results on chip design benchmarks showcase the effectiveness of our approach, with code that will be made publicly available.

1 Introduction

The scale of integrated circuits (ICs) has been enlarged dramatically, posing a challenge to the scalability of existing Electronic Design Automation (EDA) technologies. The increasing circuit density raises additional issues for VLSI placers and routers as the component size of modern VLSI design continues to drop and on-chip connectivity becomes increasingly sophisticated. Due to growing on-chip connectivity, concentrated needs and restricted resources, modern designs are prone to congestion issues and wirelength minimization, which has become a critical task at every stage of the chip design process. Accordingly, placement and routing that physically arranges the locations and the routes of nets become more crucial in modern VLSI systems.

Placement and global routing[1] are two critical and time-consuming steps in chip design. On the one hand, the components of a netlist, including macros and standard cells, are mapped to positions on the chip layout by placement, with standard cells being basic logic cells (e.g. logic gates) and macros as functional blocks (e.g. SRAMs). Moreover, the increasingly extensive use of intellectual property (IP) modules and pre-designed macro blocks makes mixed-size placement an indispensable part of physical design. The goal of placement is to optimize power, performance, and area (PPA) metrics meanwhile obeying the constraints e.g. placement density and routing congestion.

Global routing, on the other hand, creates routing channels inside a layout based on the placement assignment by connecting pins of positioned IC components to a net while adhering to technological limits. It is tightly coupled with the placement task, as an excellent placement solution can result in better chip area utilization, timing performance, and routability etc. The objective of routing is to minimize total wirelength without violating the limits of congestion and critical timing.

[1] Same to the state-of-the-art methods [1] and other works, in this paper, we address the global routing problem, while the detailed routing involves more physical constraints, and is beyond the scope of this paper.

Submitted to 36th Conference on Neural Information Processing Systems (NeurIPS 2022). Do not distribute.

Differing from the traditional learning-free solvers for placement and routing, learning-based models are recently explored and applied to (macro) placement [2, 1] and routing [3], which are mostly based on reinforcement learning (RL). Like in vision, language and other domains, an end-to-end network is often welcomed for the possibility for global optimization of the whole system, while such pure neural networks for joint placement and routing remains unresolved in literature. In particular, there are two standing issues worth further study in this paper: i) the macros are of mixed-size with varying width/height [4], bringing difficulty to the discrete grid-based placement by (current reinforcement) learning; ii) the scale of grid size in routing dataset e.g. ISPD-07 [5] can be intractable for RL as currently its action has to be designed at the local grid point level [6] rather than generate the whole routing across grids in one shot. In this paper, we aim to develop a flexible (mixed-size friendly) approach. More ambitiously, we further aim to enable a pure neural pipeline with both learnable placement net and routing net. Specifically, the placement part is fulfilled by a policy gradient based RL method for macros by considering their sizes, followed with gradient-based optimization for placing stand cells. The routing part involves a conditional generative model to finish the routing across pins at net level. The highlights of this work are:

1) We propose an RL-based model for learning mixed-size macro placement, which differs from existing learning-based placers [1, 2, 7] that often consider the macro by coarse grid-based mask. As such, the placement results are more realistic and require less post-processing to resolve collision.

2) We propose a conditional generative routing network to perform routing, in one-shot for each net. In contrast, existing RL-based solvers [6] need to perform routing step by step at grid point level within each net, in a sequential and inefficient manner. Moreover, the order for nets to route is adaptively optimized by learning instead of a pre-fixed one as performed in existing routers.

3) Combining these techniques, we develop a neural network-based pipeline for placement and routing, which to our best knowledge, is the first pure neural networks for placement and routing[2].

4) Experimental results on benchmarks show the relatively cost-efficiency (compared with RL-based routing solver [6]) and competitive performance. Source code will be made publicly available.

2 Related Work

For the inter-discipline nature of this paper, we briefly introduce the necessary background and related work to properly position our work with different communities from EDA to machine learning. Due to page limit, classical methods for placement and routing are presented in Appendix A.1.

Learning-based Solvers for Placement. Machine learning has recently been introduced which may help reduce the heuristics. [7] devises a cyclic framework between the reinforcement learning (RL) and SA modules, in which the RL module alters the relative spatial sequence between circuit components, while SA searches the solution space based on RL initialization. Following the seminal work [2] of learning sequential decision-making for macro placement, the method called DeepPlace in [1] proposes a joint learning technique for the placement and subsequent routing via RL.

Learning-based Solvers for Routing. The recent work [8] presents an attention-based RL method for obtaining pin order within each net (rather than net order in this paper which is of much larger size), followed by a classical pattern router. A DQN agent [6] is developed to decide on the routing direction on a grid graph at each step. It makes up a simple 12-element vector to represent the state of the environment. However, the model is trained on synthesized 8×8 and 16×16 grids, with no more than 50 nets that consist of 2 or 3 pins. To make the learning more scalable, a two-page preliminary work [9] formulates the routing of a net as an image-to-image translation and uses a variational auto-encoder (VAE) [10] model to generate the solution. However, the model is merely capable of handling a net with no more than three pins on a 64×64 grid. Compared with classic routing solvers, existing RL-based methods can be much more time-consuming, making the end-to-end learning of both placement and routing nets very hard. Moreover, compared with [8] learning the pin order inside a net, we try to learn the order for routing at the net level which is new in literature.

Generative Models for Placement and Routing. There are emerging works on introducing generative models for chip design. [11] proposes a generative adversarial network (GAN) [12] guided

[2]The recent work [1] also aims to achieve learning of both placement and routing, while for fast routing to achieve fast rollout for fitness evaluation, it runs a heuristic rip-up and reroute algorithm while we for the first time make the whole pipeline learnable for both placement and routing.

well generation framework to mimic experts' behavior from high-quality manually-crafted layouts. [13] adopts a GAN for generating wells and guides the placement in analog circuit layout synthesis. [14] proposes a generative model for the placement optimization of analog integrated circuit basic blocks. ThermGAN [15] treats the thermal map estimation problem as an image-generation problem using the generative model. [16] uses GAN to predict the congestion heatmap to assist classical routers. The work [17] proposes a conditional GAN to solve the multi-terminal path-finding task. In [18] the generative models are adopted to synthesize diverse layout patterns. More broadly speaking, generative models have also been recently applied in combinatorial optimization, specifically via a latent space learning and search scheme as done in CVAE-Opt [19]. In this paper, we adopt cGAN to generate routes in one shot for each net by regarding the layout to be routed as an image. To our best knowledge, no generative model has been successfully devised and applied to the pin routing problem for each net, which in fact involves complicated and constrained routing.

3 Methodology

Approach Overview. Given a netlist as input, the goal is to place the macros and standard cells on the chip canvas, ideally with a minimized overlapping areas. Based on the placement results, routing is performed in general to minimize the total wirelength while not violating the constraints. For placement, we aim to flexibly and efficiently handle the mixed-size macros via an RL scheme, and meanwhile optimize the net order for routing. For routing, we propose a conditional generative model to obtain the routes in one-shot instead of performing sequential pin connection as done in previous learning-based [6] as well as classic routers [20, 21]. The stacking RL placement network and generative routing network can be learned via gradient back propagation in an end-to-end manner, which meanwhile completes the whole task in line with the state-of-the-art learning-based placement and routing solvers [1, 2]. Note our work is pure learning based in contrast to [1] that still involves unlearnable classic routers in the whole pipeline. The pure neural architecture of our model implies our model has the potential to enjoy higher capacity by using larger model for further improvement.

3.1 Reinforcement Learning for Mixed-size Placement

A natural idea for classical placers to address the mixed-size issue for placement is to adopt a hierarchical approach based on partitioning. However, it sacrifices the solution quality since each sub-problem is solved independently. Meanwhile, DeepPlace as a learning-based placer assumes that each macro only occupies one cell in the grid graph and ignores pre-placed macros, which leads to a severe overlap issue in the final placement result. This motivates us to extend the formulation of DeepPlace by considering the real size of macros as well as initial placement information. We still adopt [22] as the CNN backbone and GCN [23] as the GNN backbone which consists of three layers that contain 16, 32 and 16 feature channels in policy network which is updated by Proximal Policy Optimization (PPO) [24], in line with the effective setting adopted in [1]. The revised elements of the Markov Decision Processes (MDPs) for mixed-size placement are defined as:

- **State s_t:** the state representation still consists of global image I portrayed the layout and netlist graph H contains detailed position of placed macros, except that the initial state of I is no longer a zero matrix $I_{n\times n} = \mathbf{O}$. Instead, our model preprocesses the positions of fixed macros in the dataset and sets I_{xy} as 1 if (x, y) has already been occupied before placement.

- **Action a_t:** the action of RL agent is to find the central of current macro, and position (x_o, y_o) is available if all points p in the region $\mathbf{R}$ satisfy $I_p = 0$, where $\mathbf{R} = \{(x, y) \mid |x - x_o| \leq \frac{h}{2}, |y - y_o| \leq \frac{w}{2}\}$ and h, w denote the height and width of the current macro respectively.

- **Reward r_t:** to further control the overlap in the final placement, the reward at the end of episode is a negative weighted sum of wirelength, routing congestion and overlapping area from the final solution: $R_E = -L_{wl} - \lambda_1 \cdot L_{cg} - \lambda_2 \cdot L_{ol}$ as weighted by λ_1 and λ_2.

3.2 Conditional Generative Learning for Pin Routing in Net

The Global Routing Grid Protocol. In global routing, the physical chip is usually divided into rectangular areas, as shown in the left part of Fig. 1. Each area is called a global routing cell (Gcell), which corresponds to a node $v_i \in V$ in the grid graph $G(V, E)$ on the right side. While each edge $e_{ij} \in E$ represents the joint boundary between abutting Gcells v_i and v_j. For edge e, its capacity c_e is the allowed maximum number of wires that can cross e, and the usage u_e is the actual number of

wires that e has been assigned. The overflow o_e denotes the amount of wires beyond c_e defined as $max(0, u_e - c_e)$. Each route conforms to the rectilinear Steiner tree structure [25].

Grid-based Conditional Generative Routing. The overall routing task is composed of numerous nets whose routing can be independent to each other. A single net consists of a series of pins placed in the nodes of the grid graph. Thus in this paper, we mainly consider the routing over multiple pins in a net. Given a single net, we formulate the one-net routing problem as the mapping from a one-net routing image x to the corresponding route layout image y, where x contains three channels: 1) the locations of pins, 2) the availability of horizontal grid edges, and 3) the availability of vertical grid edges, and one for y. As to the output, the value of each pixel of y denotes the likelihood of whether this grid point belongs to the route or not. Hence, the synthesis of routes can also be viewed as a binary classification of the pixels in y. The pixels whose likelihood is higher than the threshold are subsequently collected to form the route. In case of disconnected routes, we apply maze routing to refine the outcomes. In the initial design of our model, we do not adopt the random noise z, which mainly leads to producing fairly stochastic outputs, since the routing task barely requires stochasticity.

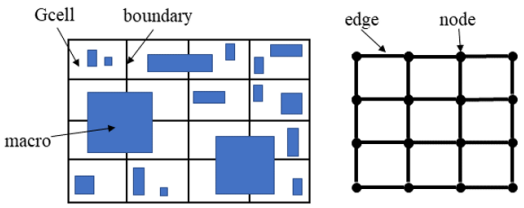


Figure 1: A chip partitioned into global cells with mixed-size macros, and its grid graph layout.

Our routing model adopts the conditional generative adversarial framework, which has shown effectiveness in image generation. The generator is composed of a basic generator for the input size of 64×64 or below (the smaller ones are padded to resize into 64×64) and an extension for the input size of larger than 64×64. The discriminator consists of two sub-discriminators to estimate routes from validity and realness. We, furthermore, design an enhanced loss to improve the performance of our model. The structure of the generative model is visualized in Fig. 2.

3.2.1 Layout Input-size-Adapting Generator

During routing, the physical chips are decomposed into Gcells in terms of various widths and heights as shown in Fig. 1, causing the diversity of the scale of the corresponding grid graphs. To make it more tractable, we develop an input-size-adapting generator to handle various grid graphs.

First, we construct a basic generator, G_{base}, to solve grids not larger than 64×64 as the chip is divided into 64×64 tiles in the macro placement stage. The architecture proposed by [26] is partly adopted as the backbone of G_{base}, which has been proven successful in generative tasks. Our basic generator contains four components : 1) a convolutional front-end, 2) a series of residual blocks, 3) a transposed convolutional component, and 4) a convolutional layer to generate the output.

Second, to handle larger grids, we establish another generator G_{large}, which is composed by two sub-networks: G_{inner} and G_{outer} (G_i and G_o, for simplicity). G_i and G_o are termed as the guiding network and the filling network, respectively. The guiding network consists of the first three parts that G_{base} owns. In contrast, the components of filling network is similar to G_{base}, and correspondingly we use $G_o^k (k = 1, 2, 3, 4)$ to denote them. We feed the input grid to G_o^1 to obtain a feature map, and downsample the input grid to feed G_i to acquire another feature map. G_o^2 takes in the element-wise sum of these two feature maps and integrates the guiding information into G_o, and the hybrid feature map then is converted into the output. The architecture of our generator is illustrated in Fig. 2(left). We can further incrementally stack additional sub-networks on G_{large}, and model compression techniques can be used to help keep the inner network neat. While training the networks, we first pre-train the sub-networks separately, and then we jointly train them to fine-tune the whole network.

Remarks. The CNN-based generator coincides with routing: 2-D neighborhood structure, translation equivariance and locality. Amid routing, chips are formulated as grids, which are further transformed into images. Routing also exhibits translation equivariance since translating a whole net with the context will not change the routing result. Moreover, for each grid node, the convolution kernel gathers the information from locally adjacent vertices, especially those directly connected to it, to form local routing features. The holistic route is then produced. Rather than simply stacking layers to handle long-range dependencies, the well-trained guiding network provides global information equivalent to long-range dependencies.

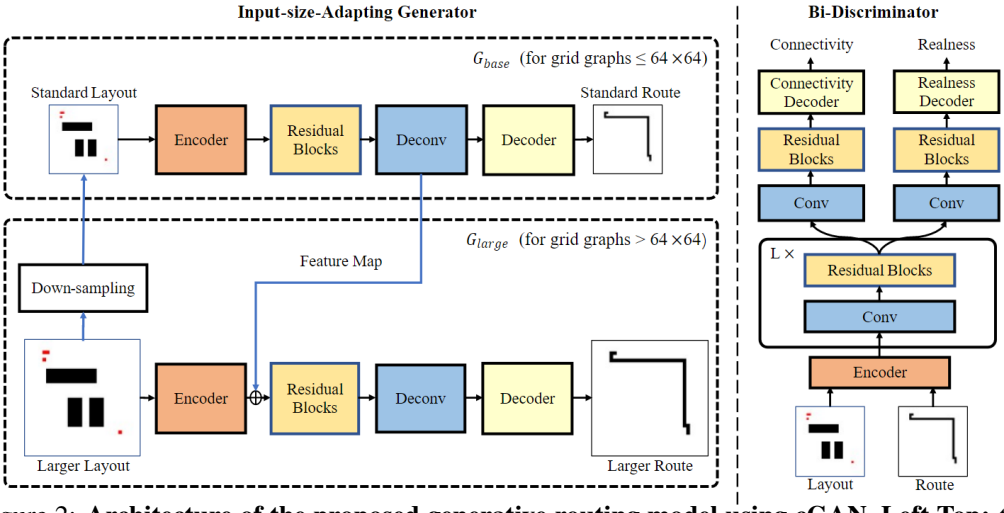


Figure 2: **Architecture of the proposed generative routing model using cGAN. Left Top:** the standard generator to handle grid graphs layout as input no bigger than 64×64. We pad black pixels to those smaller than 64×64. **Left Bottom:** the generator for larger grids. In the input layout, the red pixels represent pins, and the black blocks denote edges without capacity. We append G_{large} to G_{base}, and the two networks are jointly trained on large grids. The element-wise sum of the feature map of G_{large} and the feature map from G_{base} is fed to the residual blocks of G_{large} as the input. **Right:** The architecture of the bi-discriminator with two branches for connectivity and realness scoring. These two branches are trained with connectivity labels and realness labels, respectively.

3.2.2 Bi-Discriminator to Consider both Realness and Connectivity

Routing problems have an inherent constraint that all pins should be connected. Therefore we devise a discriminator to evaluate the connectivity of the output. To effectively train the connectivity discriminator, we develop an algorithm to accurately figure out the connectivity of each fake and real route, and then we employ the results as labels. Connectivity alone is not sufficient to evaluate the authenticity of the output, so we adopt another discriminator to estimate the realness of the output, as the original discriminator in GAN. Overall, the adversarial loss of our model can be expressed as

$$\mathcal{L}_{adv}(G, D) = \sum_{i=1,2} \lambda_i \left(\mathbb{E}_{x,y}\left[\log D_i(x, y)\right] + \mathbb{E}_x\left[\log(1 - D_i(x, G(x)))\right]\right), \tag{1}$$

where D_1 and D_2 denote the connectivity discriminator and the realness discriminator, respectively, and λ_1 and λ_2 represent the corresponding weights s.t. $\lambda_1 + \lambda_2 = 1$. The discriminators share the convolutional front-end and a stack of $L = 3$ convolutional and residual blocks, and then they make evaluations from different angles as depicted in the right part of Fig. 2.

3.2.3 Enhanced Model Loss

With the cGAN objective mixed with a traditional loss, such as L1 and L2 loss, training is inefficient as most grid points are easy negatives that cannot yield effective learning signals. In addition, tons of trivial negatives impair the training and give rise to a degraded model, and the output, thus, inclines to converge to an empty route. To bridge the gap between easy negatives and scarce positives, we apply the focal loss [27] and modify it to fit our task:

$$\mathcal{L}_{FL}(G) = -\mathbb{E}_{x,y}\left[\frac{1}{N}\sum_{i=1}^{N} \alpha\left[y_i(1 - g_i)^{\gamma}\log g_i + (1 - y_i)g_i^{\gamma}\log(1 - g_i)\right]\right], \tag{2}$$

where $i = 1, \ldots, N$ represents grid points, and y_i and g_i respectively denotes the real and generated value of corresponding grid point. We also incorporate the L2 loss into our objective to approach the real routes, and because it has been found beneficial to the synthesis [28, 29].

The introduction of the connectivity discriminator improves the correctness of the results, but at the same time, it may slightly increase the wirelength. Since the wirelength has an accurate theoretical lower bound, i.e. half-perimeter wirelength (HPWL) of the bounding box of a net, we take the difference between the length of the generated route and the HPWL as a regularization term to

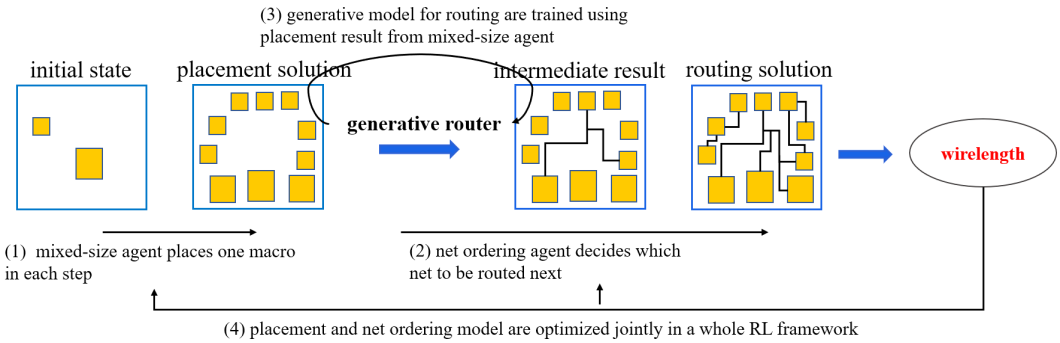


Figure 3: **Our neural macro placement and routing pipeline.** Given netlist as input, our mixed-size agent sequentially places macros on the chip layout. Generative router is then adopted to route the net chosen by net order agent. Inspired by EM algorithm, we update generative router by placement result from mixed-size agent, then placement and net order agents are learned jointly in a whole reinforcement learning framework to minimize wirelength calculated by trained generative model.

limit the wire length. We use $\mathcal{L}_r(G) = \mathbb{E}_x\left[\|l(G(x)) - h(x)\|_1\right]$ to represent the regularization term, where $l(G(x))$ denotes the length of the generated route, and $h(x)$ denotes the HPWL of the net.

The overall enhanced objective of our model gathers the above losses:

$$\min_G \left(\left(\max_D \mathcal{L}_{adv}(G, D) \right) + \mu_{FL}\mathcal{L}_{FL}(G) + \mu_{L2}\mathcal{L}_{L2}(G) + \mu_r \mathcal{L}_r(G) \right), \tag{3}$$

where μ_{FL}, μ_{L2} and μ_r are defined as the factors of $\mathcal{L}_{FL}$, $\mathcal{L}_{L2}$ and $\mathcal{L}_r$, respectively.

3.3 Neural Macro Placement and Routing Pipeline

Combining the RL-based model for learning mixed-size macro placement with one-shot generative routing network to perform routing as we introduce above, we propose a pure neural pipeline for macro placement and routing. Fig. 3 shows the flow of our mixed-size macro placer with adaptive reward function between coarse HPWL estimation to wirelength from the neural router. Given the circuit information, our mixed-size agent sequentially places the macros on the chip layout, after which the generative model for routing is adopted to connect the net chosen by net order agent and calculate wirelength as feedback. Inspired by EM algorithm, we first update the generative router using placement result from mixed-size agent (similar to **E step**), then placement and net order agents are learned jointly in a whole reinforcement learning framework to minimize wirelength calculated by trained generative model (corresponding to **M step**) following a recursive pattern.

3.3.1 Reward Adaptation between Coarse HPWL and Router's WL

For classical placers, HPWL is a common metric for estimating the true wirelength decided by routing. In our neural pipeline, however, we apply a one-shot generative routing network to route all the nets directly, which reduces bias in the reward signal. Nevertheless, it is worth noting that the untrained policy network for placement would start with random weights so that placement results are of low quality. As a result, the distribution of pins in a single net will spread out, which is difficult for a generative model-based router to produce accurate route layout images. To tackle this problem, we propose an adaptive scheme to calculate wirelength for our placement agent, integrating HPWL and neural router's output simultaneously. We introduce variable λ to scale two values and define the smoothed wirelength WL_s as follows:

$$WL_s = \lambda \cdot WL_n + (1 - \lambda) \cdot HPWL \tag{4}$$

where WL_n is the feedback of neural router. In each iteration, variable λ is updated by function $1 - e^{-0.01 \cdot n_{iter}}$. Initially, λ begins with 0 so that the wirelength is determined by $HPWL$ that serves as a coarse reward signal. As the training proceeds, the feedback of neural router gradually becomes a prominent factor to provide a more accurate objective for the placement agent.

3.3.2 Learning Net Order to Route

The order in which nets are routed is one of the most critical factors that affects the routing quality [30]. Most classical routers determine the net order by heuristics, e.g., routing smaller nets earlier [31] due to the flexibility of finding free path. However, there are diverse definitions for "smaller net", none

of which is proved to be optimal. What makes the situation worse, the complexity of real routing procedure requires us to change the net order dynamically, which is hard to implement in such a complicated system. Fortunately, our neural router divides the routing task into a series of one-net routing problems and then solves them independently, making it convenient to learn the net order.

We build net order learning module upon the neural router by developing a RL agent to determine which net to route next. Inspired by the structure of placement agent, the state of net order problem consists of routing image R as a representation of current routing layout, and graph G indicating the connectivity between nets. There are three channels of R: the locations of routed net, the capability of horizontal and vertical grid edges. Graph G is an edge-to-vertex dual of netlist graph H, whose vertices denote nets (i.e., hyperedges of H) and edges denote common cells between nets (i.e., nodes of H). Note that the state of placement and net ordering tasks are quite similar while both seek to minimize the total wirelength, we combine them into a whole RL framework by adopting same policy network to generate feature embeddings for two tasks respectively. The united structure without heuristic solver reflects the strongly coupled relationship between placement and routing, which differs from [1] that merely applies router as a black box to calculate the reward.

4 Experiments

4.1 Protocols and Setup

Experiments are conducted on a server with RTX 3090 GPUs and AMD 3970X 32-Core CPU, and implemented by PyTorch. We term our whole approach for placement and routing as **PRNet** .

Benchmarks & Datasets. For placement, we validate our RL agent for mixed-size macro placement using ISPD-2005 benchmark [32] after pre-processing, such that most fixed macros are exchanged for movable ones in line with [1]. For routing[3], we choose the ISPD-07 [5] benchmarks to produce routing instances and use the routes generated by the strong classic router [33] as training labels.

In the ISPD-07 benchmarks, some nets can contain up to hundreds of pins, but the average amount of pins in a single net is still about 4. In other words, massive nets contain no more than 4 pins. Therefore, the routing model should have sufficient ability to route the easy nets. From around 750K routing instances, we collect 30K 64×64 nets as the Route-small-4 dataset whose instance contains up to 4 pins, 80K 64×64 samples as Route-small, and 100K 128×128 samples as Route-large to evaluate the model's ability. Each of the three datasets is randomly divided into a training set (80%) and a test set (20%). These three training sets are used to train generative routing models and pick up the best one according to their performance on the test sets. Then we continue to train the best 64×64 model with additional 200K 64×64 instances and train the 128×128 model with additional 400K 128×128 instances, which is used to perform experiments on the ISPD-98 routing benchmarks [34].

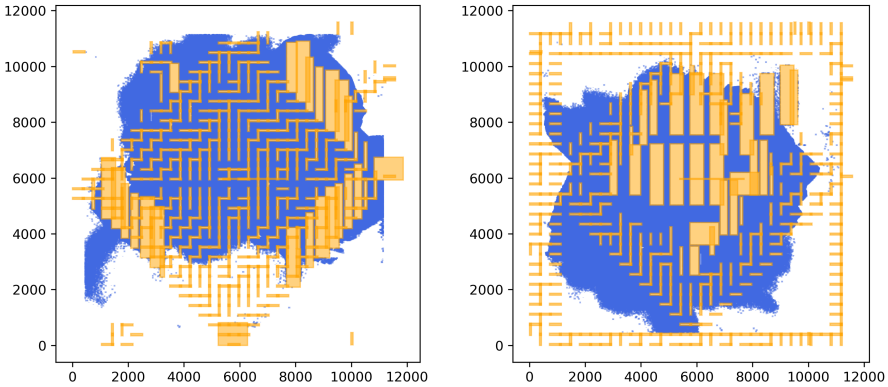
Training. To train the placement and net ordering RL agent, we use PPO [24] to update the policy network and Adam optimizer [35] is utilized with a learning rate of 2.5×10^{-4}. For training the routing models, we use Adam with learning rate of 2×10^{-4}, $\beta_1 = 0.5$, $\beta_2 = 0.999$ and a weight decay of 0.01. We employ a batch size of 64. A linear learning rate decay is also applied.

Evaluation. In mixed-size placement, we adopt HPWL as the proxy of wirelength and overlapping area to evaluate both methods while we introduce wirelength (WL) and routing congestion (RC) [36] in overall placement and routing. For routing, since there is little metric for generative routing model, we introduce the metrics correctness rate (CrrtR) and wirelength ratio (WLR) to evaluate the generated results of generative models on the datasets. CrrtR signifies the ratio of the amount of connected overflow-free routes to the number of all routes, or in short the accuracy of generated result. WLR represents the ratio of the total wirelength of connected overflow-free routes to the total wirelength of the corresponding real routes. Lower WLR indicates that the route requires fewer wires. In the experiments on the ISPD-98 routing benchmarks, wirelength, overflow and runtime are used.

4.2 Results on Mixed-size Placement

We compare the total wirelength together with overlapping area of our mixed-size approach with the state-of-the-art and open-sourced method called DeepPlace [1] as shown in Table 1. Both methods generate intermediate macro placement via RL, and then adopt gradient-based optimization placer as used in [37] to obtain complete placement solution. With only a slight increase of the total

[3] In fact we have limited choice for constructing our deep model training dataset, as there is little public dataset for training generative models for routing, and few classical routers are open-source.



(a) Result of DeepPlace [1]. (b) Result of our mixed-size placer.

Figure 4: Visualization of macro (in orange) /standard cell (in blue) placement by DeepPlace [1] and our mixed-size placer on circuit $bigblue1$. Our placer tends to place large macros in the center of canvas to avoid overlapping, while they are close to each other on the boundary for DeepPlace.

Table 1: Comparison on mixed-size placement task on the eight circuits from ISPD-2005.

Circuit	# Cells	# Mov.	Mixed-size technique (ours)		DeepPlace [1]	
			Wirelength ↓	Overlap Area↓	Wirelength ↓	Overlap Area↓
adaptec1	211K	514	82783826	**12606828**	80117232	66608273
adaptec2	255K	542	123307824	**19485631**	123265964	47085963
adaptec3	451K	710	232373680	**58588016**	241072304	140272759
adaptec4	496K	1309	234008876	**73075220**	236391936	169853555
bigblue1	278K	551	141020208	**2041890**	140435296	3519755
bigblue2	558K	948	144803296	**70702107**	140465488	103663199
bigblue3	1097K	1227	468632064	**39664931**	450633360	574956948
bigblue4	2177K	659	1001315712	**67794270**	951984128	87630042
ratio	-	-	1.000	1.0	0.987	3.9

wirelength (within 1.3% difference on average), our mixed-size macro placer achieves approximately $4\times$ reduction over DeepPlace on the overlapping area, stressing the importance of modeling macro's shape in state space. Moreover, the reduced overlapping area requires less post-processing to resolve collision, which facilitates improvement of wirelength in the long term. Examples of our mixed-size placer and DeepPlace on circuit $bigblue1$ are visualized in Fig. 4.

4.3 Results on Routing

Comparison of Generative Backbones. We compare our model with a CVAE based router using CNN as the backbone [9] and use CVAE*(CNN) to denote it. We then combine the CVAE*(CNN) with a vanilla discriminator and our bi-discriminator to build CVAE*-GAN and CVAE*-bcGAN, following the idea of [38]. In addition, we implement a U-Net based cGAN following pix2pix [29] and use cGAN(U-Net*) to denote it. Then we remove the discriminator to obtain a U-Net [39] generator and define it as U-Net*. We further replace the discriminator of cGAN(U-Net*) with our bi-discriminator. We also try to train the RL agent following the work of [6], but it fails to converge after 2-week training. Table 2 shows the results. The ResNet-based [40] models outperform the counterparts based on CVAE*(CNN) and U-Net*. Our model achieves approximately $2\times$ correctness rate, 14.7% improvement of the wirelength on Route-small-4 and 2.4% on Route-small over CVAE*(CNN). The vanilla cGAN discriminator slightly improves CVAE*(CNN) and U-Net* on one side while sacrificing the other side, and it debases the ResNet generator. However, the bi-discriminator strengthens the generators except for the CVAE*(CNN).

Ablation Studies. We conduct ablation experiments to investigate the contributions of the design choices in our model. In Table 2, we compare the full version with ResNet-based cGAN, as well as the pure ResNet generator. The ResNet generator outdoes the cGAN, but the bi-discriminator significantly improves the generator. Moreover, the enhanced loss improves the wirelength at the marginal expense of correctness. Appendix A.5.1 contains further details of comparison among loss functions, and Appendix A.5.2 shows the effectiveness of the input-size-adapting network.

Table 2: Evaluation of different backbones w.r.t. correctness rate (CrrtR) and wirelength ratio (WLR) for the routing on: Route-small-4 and Route-small. cGAN: the vanilla cGAN model with a single realness discriminator; bcGAN: the bi-discriminator version. EL: enhanced loss in Eq. 7.

our router w/ different generative models	Route-small-4		Route-small	
	CrrtR↑	WLR↓	CrrtR↑	WLR↓
CVAE*(CNN) [9]	0.414±0.020	1.179±0.033	0.397±0.008	1.042±0.006
CVAE*-cGAN(CNN)	0.557±0.065	1.292±0.108	0.439±0.021	1.315±0.015
CVAE*-bcGAN(CNN)	0.474±0.048	1.525±0.029	0.488±0.007	1.241±0.012
U-Net* [39]	0.724±0.001	3.306±0.266	0.524±0.005	1.232±0.016
cGAN(U-Net*) [29]	0.602±0.009	1.028±0.001	0.532±0.011	1.286±0.022
bcGAN(U-Net*)	0.721±0.012	1.134±0.055	0.552±0.007	1.104±0.054
ResNet [40]	0.783±0.002	1.023±0.003	0.594±0.004	1.030±0.007
cGAN(ResNet)	0.698±0.010	1.073±0.011	0.568±0.020	1.320±0.151
bcGAN(ResNet)	0.804±0.021	1.035±0.013	**0.738**±0.005	1.036±0.002
bcGAN(ResNet)+EL (full version of our router)	**0.814**±0.001	**1.010**±0.000	0.735±0.010	**1.018**±0.004

Table 3: Evaluation of wirelength (WL) and routing congestion (RC) for overall placement and routing pipeline on ISPD-05 benchmark. "GR": our generative router; "NOL": net order learning.

variants of our PRNet	adaptec1		adaptec3	
	WL↓	RC↓	WL↓	RC↓
RL-based Placer (i.e. DeepPlace [1])	6149	10.565	30154	62.751
RL-based Placer + GR	5940	10.464	29711	73.324
RL-based Placer + GR + NOL (full version of PRNet)	**5787**	**9.386**	**29462**	**43.207**

Table 4: Evaluation of wirelength (WL) and runtime (Time) with three classical routers on ISPD-98 routing benchmarks. Note that the overflow (OF) is all zero for all methods.

Circuits	Our router		NTHU-Route 2.0 [30]		BoxRouter 2.0 [41]		FastRoute 3.0 [42]	
	WL↓	Time(s)↓	WL↓	Time(s)↓	WL↓	Time(s)↓	WL↓	Time(s)↓
ibm01	**62337**	59.2	62498	1.54	62659	33	64221	0.64
ibm02	170270	179.9	**169881**	3.15	171110	36	172223	0.85
ibm03	**146362**	194.6	146458	1.49	146634	18	146753	0.49
ibm04	**165874**	254.4	166452	3.81	167275	116	170146	2.7

Test Results. We test our conditional generative routing model on the ISPD-98 benchmarks and compare the wirelength, overflow and runtime with three classical routers that perform best on ISPD-98 benchmarks. Table 9 shows the results. Our generative routing model presents competitive consequences on wirelength, while it takes a longer time to accomplish the routing task, compared with strong heuristic baselines [30, 41, 42]. Our model takes an image encoded from the whole grid with a net as the input and sequentially solve each net in a one-shot manner, while classical routers only consider the local area, which may obtain fewer wirelength yet consume more time. However, it is still much more efficient and easier to train than the RL-based router [6] from our empirical experience. We leave the speedup of our generative model for future work.

4.4 Results of Overall Placement and Routing with Ablation Study on Net Order Learning

We compare our PRNet with DeepPlace, along with an ablation study to verify the impact of net order learning. The circuits used for evaluation are the same as in mixed-size placement, and we concentrate on macros only for simplicity. Note that the real shape of macros is ignored and the grid-based mask is coarser in [1], hence the results shown in Table 3 are not identical to those in the original paper [1]. For all test cases, our neural placement and routing pipeline outperforms the other two methods in terms of both wirelength (WL) and routing congestion (RC). The significant difference in routing congestion without net order learning indicates that net order agent is able to arrange the sequence of routing efficiently, especially on circuit $adaptec3$. As a result, it is easy for every net to find free routing path while keeping away from congested area. In addition, training placement model with generative neural router in an end-to-end manner further improves the final wirelength, despite a little degradation of routing congestion if we discard the net order agent.

5 Conclusion and Discussion

We have presented a neural mixed-size placement and routing pipeline. The routing is achieved by one-shot generation of the whole path, with our devised net order learning module to dynamically adjust the routing order. Experimental results show the effectiveness of our approach.

References

[1] R. Cheng and J. Yan, "On joint learning for solving placement and routing in chip design," *NeurIPS*, 2021.

[2] A. Mirhoseini, A. Goldie, M. Yazgan, J. W. Jiang, E. Songhori, S. Wang, Y.-J. Lee, E. Johnson, O. Pathak, A. Nazi *et al.*, "A graph placement methodology for fast chip design," *Nature*, 2021.

[3] J. Liu, G. Chen, and E. F. Young, "Rest: Constructing rectilinear steiner minimum tree via reinforcement learning," in *DAC*, 2021.

[4] Y.-L. Chuang, G.-J. Nam, C. J. Alpert, Y.-W. Chang, J. Roy, and N. Viswanathan, "Design-hierarchy aware mixed-size placement for routability optimization," in *ICCAD*, 2010.

[5] G.-J. Nam, M. Yildiz, D. Pan, and P. Madden, "Ispd 2007 global routing contest," 2007.

[6] H. Liao, W. Zhang, X. Dong, B. Poczos, K. Shimada, and L. Burak Kara, "A deep reinforcement learning approach for global routing," *Journal of Mechanical Design*, 2020.

[7] D. Vashisht, H. Rampal, H. Liao, Y. Lu, D. Shanbhag, E. Fallon, and L. B. Kara, "Placement in integrated circuits using cyclic reinforcement learning and simulated annealing," *arXiv preprint arXiv:2011.07577*, 2020.

[8] H. Liao, Q. Dong, X. Dong, W. Zhang, W. Zhang, W. Qi, E. Fallon, and L. B. Kara, "Attention routing: track-assignment detailed routing using attention-based reinforcement learning," in *International Design Engineering Technical Conferences and Computers and Information in Engineering Conference*. American Society of Mechanical Engineers, 2020.

[9] D. Utyamishev and I. Partin-Vaisband, "Late breaking results: A neural network that routes ics," in *DAC*. IEEE, 2020.

[10] D. P. Kingma and M. Welling, "Auto-encoding variational bayes," in *ICLR*, 2014.

[11] B. Xu, Y. Lin, X. Tang, S. Li, L. Shen, N. Sun, and D. Z. Pan, "Wellgan: Generative-adversarial-network-guided well generation for analog/mixed-signal circuit layout," in *2019 56th ACM/IEEE Design Automation Conference (DAC)*. IEEE, 2019.

[12] I. Goodfellow, J. Pouget-Abadie, M. Mirza, B. Xu, D. Warde-Farley, S. Ozair, A. Courville, and Y. Bengio, "Generative adversarial nets," in *NIPS*, 2014.

[13] K. Zhu, H. Chen, M. Liu, X. Tang, W. Shi, N. Sun, and D. Z. Pan, "Generative-adversarial-network-guided well-aware placement for analog circuits," in *2022 27th Asia and South Pacific Design Automation Conference (ASP-DAC)*. IEEE, 2022.

[14] A. Gusmão, R. Póvoa, N. Horta, N. Lourenço, and R. Martins, "Deepplacer: A custom integrated opamp placement tool using deep models," *Applied Soft Computing*, 2022.

[15] W. Jin, S. Sadiqbatcha, J. Zhang, and S. X.-D. Tan, "Full-chip thermal map estimation for commercial multi-core cpus with generative adversarial learning** this work is supported in part by nsf grants under no. ccf-1816361, in part by nsf grant under no. ccf-2007135 and no. oise-1854276." in *2020 IEEE/ACM International Conference On Computer Aided Design (ICCAD)*. IEEE, 2020.

[16] Z. Zhou, Z. Zhu, J. Chen, Y. Ma, and A. Ivanov, "Congestion-aware global routing using deep convolutional generative adversarial networks," in *2019 ACM/IEEE 1st Workshop on Machine Learning for CAD (MLCAD)*, 2019.

[17] D. Utyamishev and I. Partin-Vaisband, "Multiterminal pathfinding in practical vlsi systems with deep neural networks," 2022.

[18] H. Yang, P. Pathak, F. Gennari, Y. C. Lai, and B. Yu, "Deepattern: Layout pattern generation with transforming convolutional auto-encoder," in *the 56th Annual Design Automation Conference 2019*, 2019.

[19] A. Hottung, B. Bhandari, and K. Tierney, "Learning a latent search space for routing problems using variational autoencoders," in *International Conference on Learning Representations*, 2020.

[20] M. D. Moffitt, "Maizerouter: Engineering an effective global router," *TCAD*, 2008.

[21] M. M. Ozdal and M. D. Wong, "Archer: A history-based global routing algorithm," *TCAD*, 2009.

[22] V. Mnih, K. Kavukcuoglu, D. Silver, A. A. Rusu, J. Veness, M. G. Bellemare, A. Graves, M. Riedmiller, A. K. Fidjeland, G. Ostrovski *et al.*, "Human-level control through deep reinforcement learning," *nature*, vol. 518, no. 7540, pp. 529–533, 2015.

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

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

[25] M. R. Garey and D. S. Johnson, "The rectilinear steiner tree problem is np-complete," *SIAM Journal on Applied Mathematics*, vol. 32, no. 4, pp. 826–834, 1977.

[26] J. Johnson, A. Alahi, and L. Fei-Fei, "Perceptual losses for real-time style transfer and super-resolution," in *European conference on computer vision*. Springer, 2016, pp. 694–711.

[27] T.-Y. Lin, P. Goyal, R. Girshick, K. He, and P. Dollár, "Focal loss for dense object detection," in *Proceedings of the IEEE international conference on computer vision*, 2017.

[28] D. Pathak, P. Krahenbuhl, J. Donahue, T. Darrell, and A. A. Efros, "Context encoders: Feature learning by inpainting," in *Proceedings of the IEEE conference on computer vision and pattern recognition*, 2016.

[29] P. Isola, J.-Y. Zhu, T. Zhou, and A. A. Efros, "Image-to-image translation with conditional adversarial networks," *CVPR 2017*, pp. 5967–5976, 2017.

[30] Y.-J. Chang, Y.-T. Lee, J.-R. Gao, P.-C. Wu, and T.-C. Wang, "Nthu-route 2.0: a robust global router for modern designs," *TCAD*, 2010.

[31] Y. Xu, Y. Zhang, and C. Chu, "Fastroute 4.0: Global router with efficient via minimization," in *ASP-DAC*. IEEE, 2009.

[32] G.-J. Nam, C. J. Alpert, P. Villarrubia, B. Winter, and M. Yildiz, "The ispd2005 placement contest and benchmark suite," in *Proceedings of the 2005 international symposium on Physical design*, 2005, pp. 216–220.

[33] W.-H. Liu, W.-C. Kao, Y.-L. Li, and K.-Y. Chao, "Nctu-gr 2.0: Multithreaded collision-aware global routing with bounded-length maze routing," *TCAD*, 2013.

[34] C. J. Alpert, "The ispd98 circuit benchmark suite," in *Proceedings of the 1998 international symposium on Physical design*, 1998, pp. 80–85.

[35] P. D. Kingma and L. J. Ba, "Adam: A method for stochastic optimization," *international conference on learning representations*, 2015.

[36] C.-K. Cheng, A. B. Kahng, I. Kang, and L. Wang, "Replace: Advancing solution quality and routability validation in global placement," *TCAD*, 2018.

[37] Y. Lin, Z. Jiang, J. Gu, W. Li, S. Dhar, H. Ren, B. Khailany, and D. Z. Pan, "Dreamplace: Deep learning toolkit-enabled gpu acceleration for modern vlsi placement," *TCAD*, 2020.

[38] X. Yu, X. Zhang, Y. Cao, and M. Xia, "Vaegan: A collaborative filtering framework based on adversarial variational autoencoders." in *IJCAI*, 2019, pp. 4206–4212.

[39] R. O, F. P, and B. T, "U-net: Convolutional networks for biomedical image segmentation," in *MICAAI*, 2015.

[40] K. He, X. Zhang, S. Ren, and J. Sun, "Deep residual learning for image recognition," in *CVPR*, 2016.

[41] M. Cho, K. Lu, K. Yuan, and D. Z. Pan, "Boxrouter 2.0: A hybrid and robust global router with layer assignment for routability," *TODAES*, 2009.

[42] Y. Zhang, Y. Xu, and C. Chu, "Fastroute3. 0: a fast and high quality global router based on virtual capacity," in *ICCAD*. IEEE, 2008.

[43] R. Kastner, E. Bozorgzadeh, and M. Sarrafzadeh, "Pattern routing: Use and theory for increasing predictability and avoiding coupling," *TCAD*, 2002.

[44] R. T. Hadsell and P. H. Madden, "Improved global routing through congestion estimation," in *Proceedings 2003. Design Automation Conference (IEEE Cat. No. 03CH37451)*. IEEE, 2003, pp. 28–31.

[45] M. Cho and D. Z. Pan, "Boxrouter: A new global router based on box expansion and progressive ilp," *TCAD*, 2007.

[46] M. A. Breuer, "A class of min-cut placement algorithms," in *DAC*, 1977.

[47] C. M. Fiduccia and R. M. Mattheyses, "A linear-time heuristic for improving network partitions," in *DAC*. IEEE, 1982.

[48] A. R. Agnihotri, S. Ono, and P. H. Madden, "Recursive bisection placement: Feng shui 5.0 implementation details," in *ISPD*, 2005.

[49] S. Kirkpatrick, C. D. Gelatt, and M. P. Vecchi, "Optimization by simulated annealing," *Science*, 1983.

[50] P. Spindler, U. Schlichtmann, and F. M. Johannes, "Kraftwerk2—a fast force-directed quadratic placement approach using an accurate net model," *TCAD*, 2008.

[51] T.-C. Chen, Z.-W. Jiang, T.-C. Hsu, H.-C. Chen, and Y.-W. Chang, "Ntuplace3: An analytical placer for large-scale mixed-size designs with preplaced blocks and density constraints," *TCAD*, 2008.

[52] A. B. Kahng and Q. Wang, "Implementation and extensibility of an analytic placer," *TCAD*, 2005.

[53] 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," *TODAES*, 2015.

[54] T. Taghavi, X. Yang, and B. Choi, "Dragon2005: Large-scale mixed-size placement tool," in *ISPD*, 2005.

[55] H. Chen, Y. Chuang, Y.-W. Chang, and Y. Chang, "Constraint graph-based macro placement for modern mixed-size circuit designs," in *ICCAD*, 2008.

[56] T.-H. Wu, A. Davoodi, and J. T. Linderoth, "Grip: Scalable 3d global routing using integer programming," in *DAC*, 2009.

[57] T.-H. Wu, A. Davoodi, and J. T, Linderoth, "A parallel integer programming approach to global routing," in *Design Automation Conference*. IEEE, 2010.

[58] C. Chu and Y.-C. Wong, "Flute: Fast lookup table based rectilinear steiner minimal tree algorithm for vlsi design," *TCAD*, 2007.

[59] C. Y. Lee, "An algorithm for path connections and its applications," *IRE Transactions on Electronic Computers*, 1961.

[60] H.-Y. Chen, C.-H. Hsu, and Y.-W. Chang, "High-performance global routing with fast overflow reduction," in *ASP-DAC*. IEEE, 2009.

Checklist

1. For all authors...
   (a) Do the main claims made in the abstract and introduction accurately reflect the paper's contributions and scope? [Yes]
   (b) Did you describe the limitations of your work? [Yes] See Appendix A.2
   (c) Did you discuss any potential negative societal impacts of your work? [Yes] See Appendix A.2
   (d) Have you read the ethics review guidelines and ensured that your paper conforms to them? [Yes]
2. If you are including theoretical results...
   (a) Did you state the full set of assumptions of all theoretical results? [N/A]
   (b) Did you include complete proofs of all theoretical results? [N/A]
3. If you ran experiments...
   (a) Did you include the code, data, and instructions needed to reproduce the main experimental results (either in the supplemental material or as a URL)? [Yes] See Section 4.1.

(b) Did you specify all the training details (e.g., data splits, hyperparameters, how they were chosen)? [Yes] See Section 4.1.

(c) Did you report error bars (e.g., with respect to the random seed after running experiments multiple times)? [Yes]

(d) Did you include the total amount of compute and the type of resources used (e.g., type of GPUs, internal cluster, or cloud provider)? [Yes] See Section 4.1.

4. If you are using existing assets (e.g., code, data, models) or curating/releasing new assets...

(a) If your work uses existing assets, did you cite the creators? [Yes]

(b) Did you mention the license of the assets? [Yes]

(c) Did you include any new assets either in the supplemental material or as a URL? [N/A]

(d) Did you discuss whether and how consent was obtained from people whose data you're using/curating? [N/A]

(e) Did you discuss whether the data you are using/curating contains personally identifiable information or offensive content? [N/A]

5. If you used crowdsourcing or conducted research with human subjects...

(a) Did you include the full text of instructions given to participants and screenshots, if applicable? [N/A]

(b) Did you describe any potential participant risks, with links to Institutional Review Board (IRB) approvals, if applicable? [N/A]

(c) Did you include the estimated hourly wage paid to participants and the total amount spent on participant compensation? [N/A]

A Appendix

In this appendix, we give more description on the classic learning-free solvers. We also give more discussion on the possible limitation of the method.

More importantly, we devise two additional methods for further empirical comparison, particularly with other (older) classic routing solvers. The results show that our generative router is indeed much more efficient than RL-based peer method which in fact is our implementation of the recent work [6]. Though our presented generative router in the main text is less efficient than state-of-the-art routers: NTHU-Route 2.0 [30], BoxRouter 2.0 [41], FastRoute 3.0 [42], while we show that our speedup version (by approximate parallelization of net routing) is comparable to extra classic routing solvers: Labyrinth 1.1 [43], Fengshui 5.1 [44] and BoxRouter [45] in runtime, on our limited computing resource (a server with RTX 3090 GPUs and AMD 3970X 32-Core CPU). We believe there is much room to improve the cost-efficiency of our generative router by more careful parallelization design and more efficient code-level implementation as current compared routers are all written and optimized in C/C++ while we use PyTorch.

We hope our appendix can help the reviewers make more informative review and decision. The source code and trained models used in the appendix will also be made publicly available.

A.1 Related Work on Classic Solvers for Placement and Routing

Classic Solvers for Placement. Global placement solvers have been studied for a long history [46, 47], as basically in three categories: partitioning-based methods, stochastic/hill-climbing methods, and analytic solvers. Partition-based approaches utilize the divide-and-conquer strategy in the early years, after which multi-level partitioning algorithms [48] are devised, with many stochastic methods using annealing [49]. When regarding to analytic solvers, force-directed algorithms [50] and non-linear optimizers [51, 52] are extensively adopted. Modern analytical placers, e.g. ePlace [53] and RePlAce [36], have recently introduced an electrostatics-based system that contains global-smooth density cost function and nonlinear optimizers, which are accelerated by DREAMPlace [37] later based on deep neural network. Mixed-size placement attracts attention for its practical value. Hierarchical [54] and constraint graph-based [55] are proposed to place large scale mixed size designs.

Classic Solvers for Routing. According to the way for solving global routing, there are two types of classic methods: concurrent and sequential solvers. Concurrent approaches attempt to handle numerous nets simultaneously. BoxRouter [45] develops progressive integer linear programming (ILP) and adaptive maze routing to diffuse the congestion. BoxRouter 2.0 [41] further provides a more systematic way of eliminating congestion and assigning layers to wires. GRIP [56, 57], built on a partitioning strategy in a 3D manner, obtains the ideal wirelength but leads to prohibitive runtime. While sequential approaches typically employ net decomposition [58], maze routing [59], pattern routing [43], or negotiation-based rip-up and rerouting (NR&R), and solely route a 2-pin net every time (e.g. [20, 21, 60, 33]). The sequential methods have been often shown faster than the concurrent approaches, but they may rely on the ordering of the nets and thus leading to sub-optimal solutions.

A.2 Limitation

The limitation refers to our end-to-end pipeline which can be a little blackbox compared with heuristic algorithms. Moreover, the conditional generative routing model is currently trained on the routes generated by the state-of-the-art classical router. Therefore, the performance of the routing model may be limited by the router. Unsupervised learning and semi-supervised learning are possible approaches to break the performance limitation.

A.3 Negative Social Impact

We have not identified potential negative social impact of our work.

A.4 Additional Methods: an RL Router and Speedup Version of Our Generative Router

To make a more comprehensive study, we further devise two additional routing methods for comparison. One is an RL-based router following [6] which only gives limited result on small-scale dataset

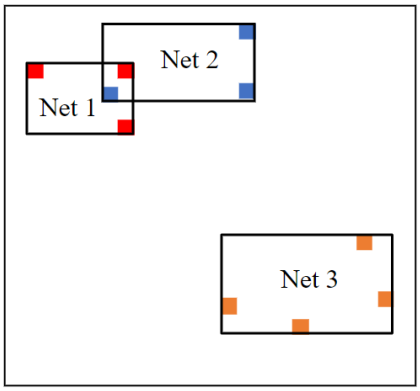


Figure 5: **An example of three nets on a chip.** Pins of $net1$, $net2$ and $net3$ are dyed red, blue and orange, respectively. The block that exactly surrounds each net is the bounding box of the net. The bounding box of $net1$ overlaps with the bounding box of $net2$, while the bounding box of $net3$ is independent of the above two.

that is generated randomly. The other is a concurrent version of our proposed generative router, to speed up the routing process.

A.4.1 Our Implemented RL Router

The RL-based router in our experiment is based on recent work [6] that decides actions of the routing direction, i.e. going north, south, etc in each step. In our implementation, we enlarge the grid size from 8×8 to 16×16 while ignoring different layers in routing, as we concentrate on calculating the final wirelength in more realistic cases. The structure of our Q-network consists of three fully-connected layers with 128 hidden units in each layer. Reward is set as -1 for all valid actions, except that the reward of finding terminal pin is 50 and a penalty of -50 is supposed to pay for invalid action which makes the capacity of routing path smaller than zero.

A.4.2 Speedup Version of Our Generative Router

In the main text, our generative router routes the nets sequentially (i.e. one net by one net) such that it produces routes with lower wirelength and overflow. In another word, the batch size in the inference stage for the deep routing network is 1 which is not common and does not release the computing advantage of GPU. To speed up our generative router, we set the batch size to 128 in the inference stage to enable our router to run in a concurrent manner.

However, the concurrent version may degenerate the performance of the outputs on wirelength and overflow. As illustrated in Fig. 5, the bounding box of $net1$ overlaps with that of $net2$, and the bounding box of $net3$ is far away from them. When routing these three nets concurrently, the routes of $net1$ and $net2$ may intersect with each other and cause conflict in the usage of routing capacity, which leads to overflow and makes routing harder for the rest of nets.

A.5 Additional Results

A.5.1 Ablation Study for Loss Function of the Routing Model

In this experiment, we utilize our conditional generative routing network as the base model and verify the effectiveness of the enhanced loss function. Our adversarial loss is

$$\mathcal{L}_{adv}(G, D) = \sum_{i=1,2} \lambda_i \left(\mathbb{E}_{x,y}\left[\log D_i(x, y)\right] + \mathbb{E}_x\left[\log(1 - D_i(x, G(x)))\right]\right), \tag{5}$$

where we set $\lambda_1 = \lambda_2 = 0.5$ by default. While our focal loss is

$$\mathcal{L}_{FL}(G) = -\mathbb{E}_{x,y}\left[\frac{1}{N}\sum_{i=1}^{N} \alpha\left[y_i(1 - g_i)^{\gamma} \log g_i + (1 - y_i) g_i^{\gamma} \log(1 - g_i)\right]\right], \tag{6}$$

Table 5: Loss function comparison w.r.t Correctness Rate (CrrtR) and Wirelength Ratio (WLR) on Route-small dataset (80K samples). We use our conditional generative network as the base model to conduct this experiment. "L2" is short for L2 loss, "FL" is short for focal loss, and "EL" denotes the enhanced loss in Eq. 7.

Loss	Route-small	
	CrrtR↑	WLR↓
L2	0.648	1.020
FL	0.756	1.091
L2+FL	0.674	1.035
EL	0.735	1.018

Table 6: Comparison w.r.t Correctness Rate (CrrtR) and Wirelength Ratio (WLR) on Route-large dataset (100K 128×128 samples). "Non-ISA" is the non-input-size-adapting network. "ISA" is the input-size-adapting network, and "0", "50" and "100" refer to the amount of pre-training epochs for the filling network.

Variants	Params	Route-large	
		CrrtR↑	WLR↓
Non-ISA	15M	0.657	1.023
ISA-0	12M	0.777	1.011
ISA-50	12M	0.784	1.010
ISA-100	12M	0.780	1.013

where we set $\alpha = 0.5$ and $\gamma = 2$ by default. For the enhanced loss, it can be expressed as

$$\min_{G} \left(\left(\max_{D} \mathcal{L}_{adv}(G, D) \right) + \mu_{FL}\mathcal{L}_{FL}(G) + \mu_{L2}\mathcal{L}_{L2}(G) + \mu_{r}\mathcal{L}_{r}(G) \right), \tag{7}$$

where we set $\mu_{FL} = \mu_{L2} = 5 \times 10^3$ and $\mu_r = 0.1$ in all our experiments.

In Table 5, focal loss delivers the best correctness rate while introducing higher wirelength. In contrast, the L2 loss is opposite of the focal loss. The enhanced loss combines the advantages of these two loss functions and obtains a competitive correctness rate and the best wirelength ratio.

A.5.2 Input-size-Adapting vs. Non-Input-size-Adapting

We explore the performance difference between our input-size-adapting network (represented by ISA) and a non-input-size-adapting variant (represented by Non-ISA). The Non-ISA follows the structure of our basic generator. Its convolutional front-end contains one more layer, and it has three additional residual blocks. The ISA utilizes the well-trained basic generator in part as the guiding network to obtain a feature map, and it further uses another convolutional front-end as the filling network to acquire another feature map. Then the element-wise sum of the feature maps are fed into a series of residual blocks and a decoder to generate the output. We compare the Non-ISA with 3 variants of our ISA (without pre-training the filling network, pre-training the filling network for 50 epochs and pre-training the filling network for 100 epochs). The ISA outperforms the Non-ISA as illustrated in Table 6, while the variants pre-trained for different number of epochs behave similarly.

Fig. 6 shows the training plot for the Non-ISA, the ISA and the basic generator. The variants of the ISA pre-trained for diverse amounts of epochs share similar training curves. The training of all the networks is stable, and the ISA converges faster than the Non-ISA and achieves a better result since it partly inherits the well-trained basic generator and the filling network has also been pre-trained.

A.5.3 Additional Experiments of Generative Backbones

We extend the dataset Route-small to a larger dataset named Route-small-extension that contains 650k instances from 3 circuits as the training set and 200k samples from another circuit as the test set. We conduct additional experiments of generative routing models on Route-small-extension to make

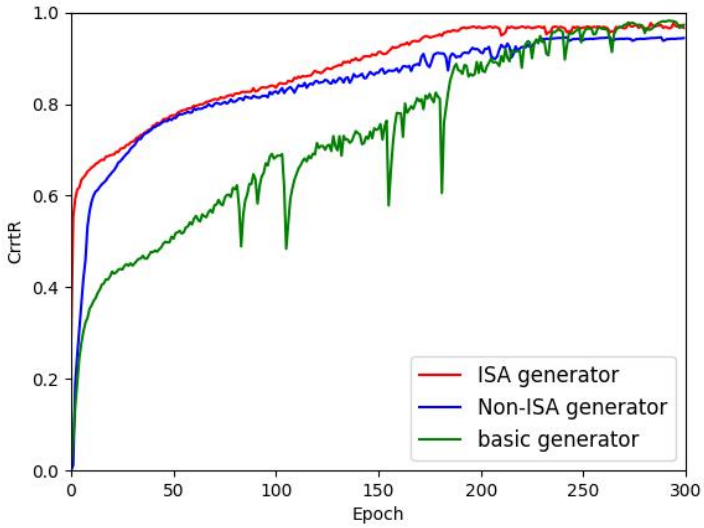


Figure 6: Training curves of the Non-ISA generator, the ISA generator and the basic generator. The variants of the ISA pre-trained for different amounts of epochs hold similar training curves.

Table 7: Evaluation of different backbones w.r.t. correctness rate (CrrtR) and wirelength ratio (WLR) for the routing on: Route-small-extension. cGAN: the vanilla cGAN model with a single realness discriminator; bcGAN: the bi-discriminator version. EL: enhanced loss in Eq. 7.

our router w/ different generative models	Route-small–extension	
	CrrtR↑	WLR↓
CVAE*(CNN) [9]	0.597	1.073
CVAE*-cGAN(CNN)	0.535	1.081
CVAE*-bcGAN(CNN)	0.647	1.023
U-Net* [39]	0.728	1.014
cGAN(U-Net*) [29]	0.572	1.077
bcGAN(U-Net*)	0.731	1.022
ResNet [40]	0.743	1.273
cGAN(ResNet)	0.632	1.017
bcGAN(ResNet)	0.757	1.010
bcGAN(ResNet)+EL (full version of our router)	**0.795**	**1.007**

Table 8: Comparison w.r.t wirelength (WL), overflow (OF) and runtime (time) for our conditional generative routing model and RL-based router on circuit $adaptec1$ from ISPD-2005 benchmark. Note the maximal grid size that RL router can deal with is 16×16, so the positions of macros for both methods are scaled to $[0, 16)$ for a fair comparison.

Method	WL↓	OF ↓	Time(s)↓
Generative Router (ours)	5240	0	**1.805**
RL Router	4951	157	11.528

further comparison among the generative backbones. The results are shown in Table 7, which is an extension to Table 2.

A.5.4 Comparison with RL-based Router

Table 8 compares the results between our conditional generative routing model and RL-based router on circuit $adaptec1$ from ISPD-2005 benchmark. Our conditional generative router is much more efficient than the RL-based router, achieving approximately $10\times$ speedup. In addition, the overflow of RL-based router is greater than zero, which degrades the performance of wirelength in the long run. Note the maximal grid size that RL router can deal with is 16×16 while our generative routing model is still applicable for grids larger than 64×64. Such limitation on size and extremely low efficiency make RL-based router not an option in neural macro placement and routing pipeline.

Table 9: Evaluation of wirelength (WL), overflow (OF) and runtime (Time) with three classical routers on ISPD-98 routing benchmarks. Sequential denotes the variant of our router sequentially routing each net, while Concurrent represents the concurrently routing variant.

Circuits	Our router (Sequential)			Our router (Concurrent)			Labyrinth 1.1 [43]			Fengshui 5.1 [44]			BoxRouter [45]		
	WL↓	OF↓	Time(s)↓	WL↓	OF↓	Time(s)↓	WL↓	OF↓	Time(s)↓	WL↓	OF↓	Time(s)↓	WL↓	OF↓	Time(s)↓
ibm01	**62337**	0	59.2	64094	43	17.6	76517	398	21.2	66006	189	15.1	65588	102	8.3
ibm02	**170270**	0	179.9	173391	25	27.1	204734	492	34.5	178892	64	47.9	178759	33	34.1
ibm03	**146362**	0	194.6	148629	5	24.4	185116	209	36.3	152392	10	35.2	151299	0	16.9
ibm04	**165874**	0	254.4	168897	142	36.1	196920	882	83.5	173241	465	54.1	173289	309	23.9
ibm05	**408421**	0	189.9	409356	4	43.7	420583	0	59.2	412197	0	104.8	409747	0	49.5
ibm06	**278029**	0	232.9	279664	27	38.2	346137	834	104.3	289276	35	80.1	282325	0	33.0

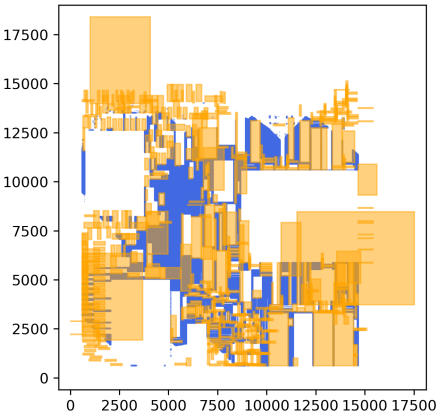
A.5.5 Comparison with a Concurrent Version of Our Router and Other Classic Routers

We first test our conditional generative routing model with each net routed sequentially (i.e. a batch size of 1) on the ISPD-98 benchmarks. In addition, we run our generative router with nets routed concurrently (a batch size of 16), which may trade wirelength and overflow for runtime, and we reimplement the post-processing module by C++. The sequential and concurrent variants of our generative routing model are defined as Sequential and Concurrent, respectively. We further compare the wirelength, overflow and runtime with three more classical routers, Labyrinth 1.1 [43], Fengshui 5.1 [44] and BoxRouter [45]. Since these three routers are not open-source and are difficult to reproduce, we directly use the experimental results in their papers. The experiments of the classical routers are performed on a 1.4GHz PentiumIII workstation with Linux operating system. Our experiments are performed on a server with RTX 3090 GPUs and AMD 3970X 32-Core CPU.

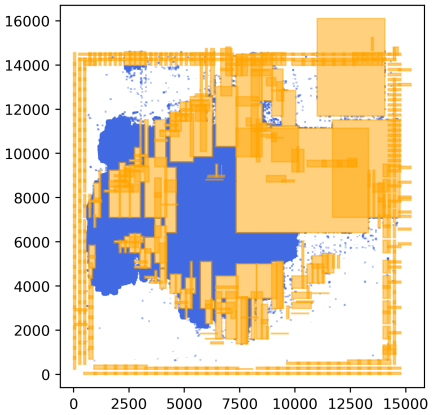
Table 9 shows the results. On the one hand, our sequential generative routing model outperforms the classical routers on wirelength and overflow, while it takes a longer time to accomplish the routing task, compared with strong heuristic baselines. On the other hand, our concurrent version dramatically speeds up the running ($3.36 \sim 7.98\times$ speedup) of generative routing model at the expense of total wirelength and overflow, and it still performs better than the three classical routers on wirelength and overflow, except the overflow on $ibm05$ and the overflow of BoxRouter on $ibm03$ and $ibm06$. Modifying our algorithms and finding other techniques to cut down the runtime are still being explored.

A.5.6 Additional Visualization of Mixed-size Placement

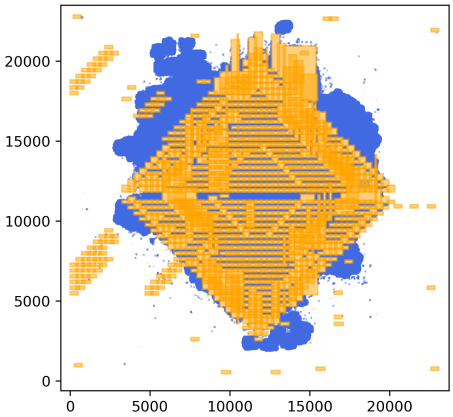
Despite circuit $bigblue1$, two additional visualization of our mixed-size placer and DeepPlace on circuits $adaptec2$ and $adaptec4$ are elaborated in Fig. 7.



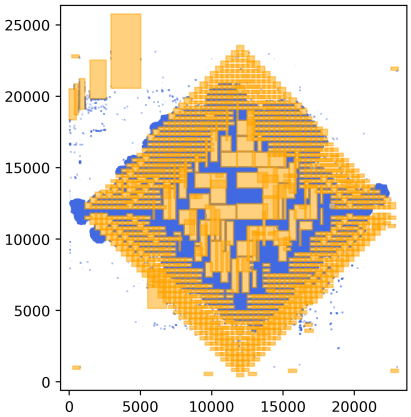
(a) DeepPlace on $adaptec2$.



(b) Our mixed-size placer on $adaptec2$.



(c) DeepPlace on $adaptec4$.



(d) Our mixed-size placer on $adaptec4$.

Figure 7: Visualization of macro (in orange) /standard cell (in blue) placement by DeepPlace [1] and our mixed-size placer on circuits $adaptec2$ and $adaptec4$. On circuit $adaptec4$, the density of macros makes it difficult to fill standard cells into the center of canvas for DeepPlace, while our mixed-size placer eliminates the problem via reducing overlap.