ChiPFormer: Transferable Chip Placement via Offline Decision Transformer

Yao Lai [1] **Jinxin Liu** [4] **Zhentao Tang** [3] **Bin Wang** [3] **Jianye Hao** [3 5] **Ping Luo** [2 1]

Abstract

Placement is a critical step in modern chip design, aiming to determine the positions of circuit modules on the chip canvas. Recent works have shown that reinforcement learning (RL) can improve human performance in chip placement. However, such an RL-based approach suffers from long training time and low transfer ability in unseen chip circuits. To resolve these challenges, we cast the chip placement as an offline RL formulation and present ChiPFormer that enables learning a transferable placement policy from fixed offline data. ChiPFormer has several advantages that prior arts do not have. First, ChiPFormer can exploit offline placement designs to learn transferable policies more efficiently in a multi-task setting. Second, ChiPFormer can promote effective finetuning for unseen chip circuits, reducing the placement runtime from hours to minutes. Third, extensive experiments on 32 chip circuits demonstrate that ChiPFormer achieves significantly better placement quality while reducing the runtime by 10× compared to recent state-of-the-art approaches in both public benchmarks and realistic industrial tasks. The deliverables are released at sites.google.com/view/chipformer/home.

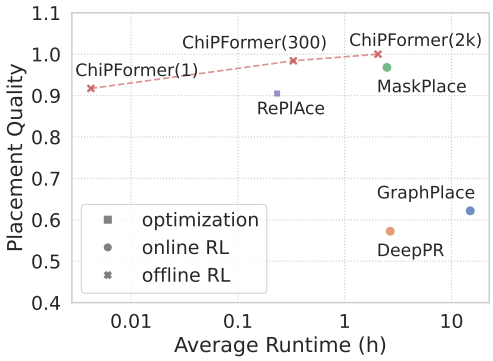


Figure 1: **Comparing the placement quality and the runtime** between ChiPFormer (ours) and the recent advanced RL-based methods such as GraphPlace (Mirhoseini et al., 2021), MaskPlace (Lai et al., 2022), and DeepPR (Cheng & Yan, 2021) and optimization method RePlAce (Cheng et al., 2018). All methods are evaluated in the *ISPD05* benchmark. The placement quality (higher is better) is normalized by $1.25^{(1-\text{HPWL}/\min \text{HPWL})} \in [0, 1]$, where HPWL represents Half Perimeter Wire Length. The number inside "ChiPFormer(·)" means the maximum few-shot number for placement in the finetuning stage (*i.e.,* rollout times). We see that ChiPFormer is the first offline RL approach for chip placement so far, and ChiPFormer(300) outperforms the other baselines in terms of quality and efficiency.

1. Introduction

In modern chip design, placement is a crucial and challenging problem, which places circuit modules (*e.g.,* macros and standard cells) with varying sizes on a chip canvas. The placement result can determine a chip's performance, such as the speed and energy cost, especially when the scale of integrated circuits grows continuously. For example, very large-scale integrated (VLSI) circuits can have 100 to 1k macros (*e.g.,* SRAMs, IOs, and packaged computing units) and 10k to 100k standard cells (*e.g.,* logical gates), making the placement problem computationally expensive.

Recent advanced approaches (Mirhoseini et al., 2021; Cheng & Yan, 2021; Lai et al., 2022) have shown that reinforcement learning (RL) can produce chip layouts that are superior or comparable to those designed by humans in many key evaluation metrics such as wirelength and congestion while spending less placement time than humans. Specifically, these approaches typically treat the placement problem as a Markov Decision Process (MDP) and place one circuit module at each step. All of them learn the placement policy in an online manner by iteratively collecting data through interactions with the environment, as shown in Fig.2 (a).

[1] Department of Computer Science, The University of Hong Kong, Hong Kong [2] Shanghai AI Laboratory, China [3] Huawei Noah's Ark Lab, China [4] Zhejiang University, China [5] Tianjin University, China . Correspondence to: Ping Luo <pluo@cs.hku.hk>, Jianye Hao <jianye.hao@tju.edu.cn>.

Proceedings of the 40^{th} International Conference on Machine Learning, Honolulu, Hawaii, USA. PMLR 202, 2023.

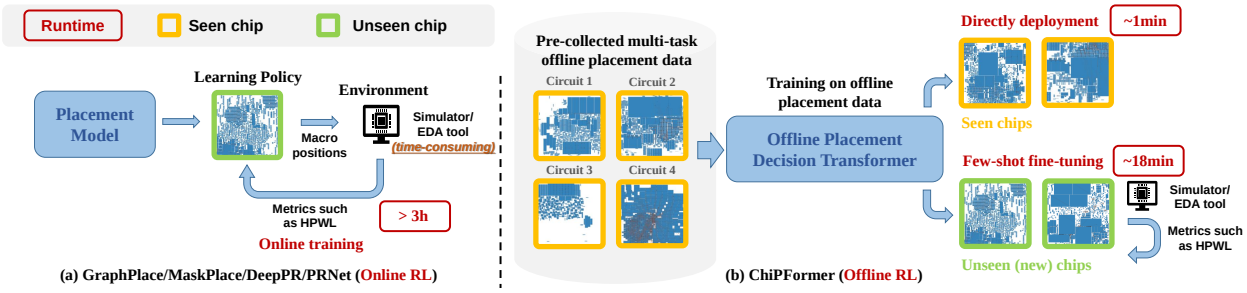


Figure 2: **Comparing the overall pipelines between (a) online RL placement and (b) offline RL placement (ours).** In (a), the online RL model keeps interacting with the environment (*i.e.,* a placement simulator or an EDA design tool for getting metrics from placement designs, and its time consumption is usually proportional to the circuit scale) to learn policy from scratch. As a result, online policy learning takes more than 3 hours on a chip in the *ISPD05* benchmark. In (b), the offline approach enables ChiPFormer to learn a policy using fixed offline placement data, thus eliminating time-consuming online interactions. When an *unseen* circuit is presented, ChiPFormer can be finetuned with only a few rollouts by reusing and transferring the learned experience across multi-task offline data. We see that such an offline method can reduce the runtime in the *ISPD05* benchmark ten times compared to previous online approaches (*i.e.,* 18 minutes *v.s.* 3 hours).

However, one challenge that remains unsolved is that all recent RL methods have a low training efficiency, as shown in Fig.1. This is because learning the policy online is slow in a large placement search space, significantly when the chip circuit scale increases. For instance, Mirhoseini et al. (2021) pointed out that the search space can be larger than 10^{2500} when there are only 1000 modules. In that case, it takes approximately 48 hours to pretrain a policy network and 6 hours to finetune it on eight V100 GPUs. Although DeepPR (Cheng & Yan, 2021) and MaskPlace (Lai et al., 2022) can shorten the runtime, they still require more than 3 hours of training. RL-based methods can achieve much stronger placement performance than the classic optimization approaches (Lin et al., 2020; Cheng et al., 2018), but their long runtime makes them less practical than classic methods that can produce a placement design in minutes.

This work addresses the above challenge by presenting ChiPFormer, which designs a decision transformer model for chip placement. It has three appealing benefits compared to the recent approaches. First, unlike the current advanced RL methods that learn the placement policy online, ChiPFormer formulates chip placement as an offline RL problem in a data-driven manner. This offline formulation enables us to pretrain ChiPFormer on fixed and pre-collected data, alleviating the time-consuming online rollouts and enabling data reuse across multiple placement tasks. The learned policy can be transferred to unseen chip circuits in a few minutes.

Second, ChiPFormer can learn transferable placement policy, which is achieved by collecting multi-task offline data on multiple circuits and then modeling a conditional placement policy with ChiPFormer. Following Gato (Reed et al., 2022), we assume the offline data are collected from multiple chip circuits using (near) expert-level placement behaviors, differing from the common offline RL setting where the data are collected by sub-optimal behavior policies over a single environment such as Chen et al. (2021).

Third, unlike the recent representative RL methods such as GraphPlace (Mirhoseini et al., 2021), MaskPlace (Lai et al., 2022), and DeepPR (Cheng & Yan, 2021) that learned the placement policy using convolutional neural network (CNN) or graph neural network (GCN), ChiPFormer is the first transformer placement network so far. Our multi-task transformer design allows us to learn transferable policies, which can be generalized to new unseen circuits within a few minutes. For example, we apply ChiPFormer on 12 unseen chip circuits and achieve an average placement time of 18 minutes, outperforming the GraphPlace method (Mirhoseini et al., 2021) in HPWL and runtime metrics by 65% and 97% decrease, respectively.

This paper has three main **contributions**. (1) To our knowledge, ChiPFormer is the first work so far to learn transferable placement policy in an offline RL manner. The learned policy can generalize to unseen chip placement tasks effectively and efficiently. (2) To facilitate further study of the offline placement problem, we release the collected placement dataset, including 12 chip circuits (tasks) and 500 expert placement results for each circuit.[1] (3) We conduct extensive experiments on 32 circuit tasks, containing 26 circuits from public chip benchmarks and 6 circuits from realistic industrial chips. This work has evaluated $1.3\times \sim 5.3\times$ more chip circuits than recent works. For example, there are 6 circuits evaluated in GraphPlace, 24 in MaskPlace, and 8 in DeepPR. In all experiments, ChiPFormer can speed

[1]The dataset is shared on Google drive.

up the placement runtime by $10\times \sim 30\times$ with only two Nvidia 3090 GPUs while achieving better placement quality compared to the recent state-of-the-art methods.

2. Preliminaries

Chip Placement. The chip placement problem can be defined as a constrained optimization problem. The main objective is to determine the positions of circuit modules (*e.g.,* macros and standard cells) on a physical chip canvas to minimize the wirelength, which determines the chip delay and energy consumption. As the computation of wirelength is an NP-complete problem (Garey & Johnson, 1977), recent placement methods use Half-Perimeter Wire-Length (HPWL) as a proxy to estimate the wirelength, which is computed by accumulating all the half-perimeters of bounding boxes of all the nets from the circuit netlist.

As shown below, the constraints for placement include: (1) *overlap constraint*, which avoids the overlapping between modules (*i.e.,* each position on the chip canvas can be covered by at most one module), and (2) *congestion constraint*, where the wire congestion should be lower than a desired small threshold to reduce chip cost. In general, the placement optimization problem can be formulated as

$$\min_{\boldsymbol{x},\boldsymbol{y}} \text{HPWL}(\boldsymbol{x},\boldsymbol{y}), \tag{1}$$

$$\text{s.t. Overlap}(\boldsymbol{x},\boldsymbol{y},\boldsymbol{w},\boldsymbol{h}) = 0, \tag{2}$$

$$\text{Congestion}(\boldsymbol{x},\boldsymbol{y},\boldsymbol{w},\boldsymbol{h}) \leq C, \tag{3}$$

where $(\boldsymbol{x},\boldsymbol{y}) = (x_1, x_2, ..., x_n, y_1, y_2, ..., y_n)$ and each pair (x_i, y_i) is the placement position of the i^{th} circuit module. Similarly, $(\boldsymbol{w},\boldsymbol{h})$ are the widths and heights of modules. C is the desired threshold of the congestion constraint. HPWL$(\cdot)$, Overlap$(\cdot)$, and Congestion$(\cdot)$ are the functions to calculate the HPWL, the overlap area, and the congestion of the placement design, respectively.

Offline Reinforcement Learning. RL is typically designed to deal with tasks of sequential modeling, which is often described by the Markov Decision Process (MDP). Particularly, an MDP, written as $\mathcal{M} := (\mathcal{S}, \mathcal{A}, \mathcal{T}, r, \rho, \gamma)$, is specified by the state space $\mathcal{S}$, action space $\mathcal{A}$, transition dynamics $\mathcal{T}(\boldsymbol{s}_{t+1}|\boldsymbol{s}_t, \boldsymbol{a}_t)$, reward function $r(\boldsymbol{s}_t, \boldsymbol{a}_t)$, initial state distribution $\rho(\boldsymbol{s}_1)$, and discount factor γ. The goal of RL is to learn a policy $\pi(\boldsymbol{a}|\boldsymbol{s})$ that maximizes the expected return $\mathbb{E}_{\boldsymbol{\tau}\sim\pi(\boldsymbol{\tau})}[\sum_{t=1}^{T} r(\boldsymbol{s}_t, \boldsymbol{a}_t)]$, where $\boldsymbol{\tau} := (\boldsymbol{s}_1, \boldsymbol{a}_1, ..., \boldsymbol{s}_T, \boldsymbol{a}_T)$ denotes the state-action trajectory. For clarity, we employ the notation $\pi(\boldsymbol{\tau})$ to denote the trajectory distribution induced by executing the policy $\pi(\boldsymbol{a}|\boldsymbol{s})$ in the environment.

In *offline* RL, the agent (learning policy) will not interact with the environment (*i.e.,* it cannot input the placement design into the simulator/EDA tool and get the metrics). Instead, the agent is provided with a fixed dataset $\mathcal{D} := \{(\boldsymbol{s}, \boldsymbol{a}, \boldsymbol{s}', r)\}$, which has been collected by some data-generating process. Since it cannot explore the MDP, the agent must rely on the provided offline data to learn a policy that maximizes the expected return.

Reinforcement Learning for Placement. To instantiate an RL-based chip placement, we can re-frame the placement as a sequential decision-making MDP, denoted by $\mathcal{M}^c$, where we use the superscript to indicate the chip placement task for a specific circuit c (Mirhoseini et al., 2021; Lai et al., 2022). In the MDP, state $\boldsymbol{s}$ describes the positions of previous macros that have been placed, action $\boldsymbol{a}$ indicates the position of the current macro to be placed, and reward r is defined as the negative wirelength and constraints (circuit-dependent).

3. Our Approach

3.1. Problem Setup

This paper studies the chip placement problem in the offline RL regime, as shown in Fig.2 (b). In particular, we cast the chip placement as a sequential decision-making problem but assume that the RL experience is fixed and there is no further interaction with the environment, which has a vast search space. In contrast, previous RL placement methods learn from scratch through expensive and time-consuming online interaction with the environment, creating a barrier to applying RL methods to efficient placement tasks. Hence, we expect that offline training would facilitate more efficient chip placement, especially when transferring to new unseen chip circuit tasks.

Formally, we have an *expert-level* and *multi-task* chip placement dataset, denoted as $\mathcal{D} := \{(c, \boldsymbol{\tau})\}$, where c denotes the index of a chip circuit task and $\boldsymbol{\tau}$ denotes the collected expert-level placement behaviors corresponding to c. Unlike the vanilla offline RL, we drop the reward term by considering expert-level behaviors.

Different from typical offline RL that assumes all offline data comes from a single task, our offline data $\mathcal{D}$ are collected from n different circuit placement tasks $\mathcal{M}_{\text{train}}^{[n]}$, where $\mathcal{M}_{\text{train}}^{[n]}$ denotes a set of MDPs $\{\mathcal{M}_{\text{train}}^{1}, \cdots, \mathcal{M}_{\text{train}}^{n}\}$. This is conceptually similar to the multi-task learning setup. In modern chip placement, the expert-level and multi-task setup can be naturally satisfied. For example, one can employ any RL-based and optimization-based methods to collect the placement data from multiple existing circuits. To facilitate future research, we have released our collected offline dataset.

3.2. Overall Pipeline

Our pipeline consists of two steps, including macro placement and mixed-size placement.

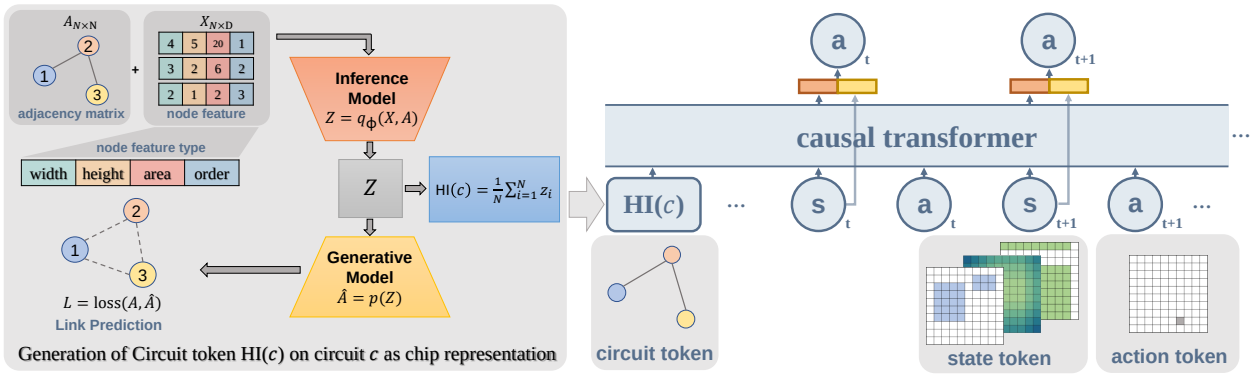


Figure 3: **Circuit token generation (left) and ChiPFormer architecture (right).** We generate the circuit token HI(c) by pretraining a conditional VAE model (graph implementation), which fits an inference model and a generative model. Specifically, the inference model takes as input the adjacency matrix $\mathbf{A}$ and node feature matrix $\mathbf{X}$ of a graph representing the topology structure of circuit c. We use the GPT architecture as the backbone of ChiPFormer, which uses a causal self-attention mask and predicts actions by feeding tokens (including circuit token, state token, and action token) autoregressively.

In the first step for macro placement, we train a general decision/sequence generation model, ChiPFormer, using only the collected data $\mathcal{D}$ of multiple placement tasks. Then, we transfer the learned policy to a downstream task (chip), denoted by $\mathcal{M}_{\text{test}}$. For example, as Fig.2 (b) shows, given an already seen chip, *i.e.,* the downstream task stays in the offline training distribution $\mathcal{M}_{\text{test}} \in \mathcal{M}_{\text{train}}^{[n]}$, we can directly use the pretrained ChiPFormer to generate a new placement design. Alternatively, given a new unseen chip, *i.e.,* $\mathcal{M}_{\text{test}} \notin \mathcal{M}_{\text{train}}^{[n]}$, we can finetune the pretrained ChiPFormer for adapting to unseen circuits with few-shot online interaction (Section 3.4). In the second step for mixed-size placement, we let the macro placement result as an initial layout status and adopt the optimized-based method to place the standard cells (Section 3.5) by following recent works such as Flora (Liu et al., 2022a) and GraphPlanner (Liu et al., 2022b).

3.3. Offline Learning of ChiPFormer

ChiPFormer models the sequential placement task as a general hindsight information matching (Furuta et al., 2021), by following prior supervised offline RL methods (Chen et al., 2021; Emmons et al., 2021). We first describe its objective and then present the details of the network architecture for sequential placement modeling.

Hindsight Information Matching. Specifically, given a trajectory $\boldsymbol{\tau} := (\boldsymbol{s}_1, \boldsymbol{a}_1, \cdots, \boldsymbol{s}_T, \boldsymbol{a}_T)$ starting from state $\boldsymbol{s}_1$, we can define *hindsight information* as a trajectory's information statistics, denoted as HI($\boldsymbol{\tau}$), which could be any function of a trajectory that captures some statistical properties in the state space or the trajectory space. For example, a prior work, Decision Transformer (Chen et al., 2021), takes the return of a trajectory as the hindsight information $\text{HI}(\boldsymbol{\tau}) := \sum_{t=1}^{T} r(\boldsymbol{s}_t, \boldsymbol{a}_t)$. The supervised goal-reaching RL (Ghosh et al., 2019) takes the final state as the hindsight information $\text{HI}(\boldsymbol{\tau}) := \boldsymbol{s}_T$. In contrast, we extend the naive hindsight information to the *multi-task* scenario by incorporating the additional task description (*i.e.,* circuit c). Instead of using heuristic statistics (like the returns), we take a learned embedding of the circuit c as the multi-task hindsight information $\text{HI}(c, \boldsymbol{\tau})$.

We then model the above hindsight information matching via supervised regression by learning a conditional placement policy π_θ,

$$\max_{\theta} \mathbb{E}_{(c,\boldsymbol{\tau})\sim\mathcal{D}} \left[\sum_{t=1}^{T} \log \pi_\theta(\boldsymbol{a}_t | \boldsymbol{\tau}_t, \text{HI}(c, \boldsymbol{\tau})) \right], \quad (4)$$

where $\boldsymbol{\tau}_t := (\boldsymbol{s}_1, \boldsymbol{a}_1, \cdots, \boldsymbol{s}_t)$ denotes the trajectory segment until the state $\boldsymbol{s}_t$. Next, we discuss how such placement hindsight information $\text{HI}(c, \boldsymbol{\tau})$ can be captured by exploiting the circuit netlist structure as its representation.

Chip Representation. Since the physical connectivity of the circuit netlist can often be represented by a graph, we use the variational graph auto-encoders (VGAE) (Kipf & Welling, 2016b) to encode the topology information of chip circuits. As shown in the left of Fig.3, we transform the netlist of the circuit c into an undirected graph $\mathcal{G}^c = (\mathcal{V}^c, \mathcal{E}^c)$, where $\mathcal{V}^c$ denotes a set of nodes representing all modules on a circuit c, and $\mathcal{E}^c$ denotes a set of edges representing the wire connecting the modules. We also define an $N \times N$ matrix $\mathbf{A}^c$ and an $N \times D$ matrix $\mathbf{X}^c$ to be the adjacency matrix and the node features of graph $\mathcal{G}^c$, respectively. We have $N = |\mathcal{V}^c|$, the number of nodes, and D, the size of the node features.

Then, we can optimize the following variational lower bound with respect to the inference parameters ϕ as shown below. For simplicity of notation, here we drop the superscript c for matrices $\mathbf{X}^c$, $\mathbf{A}^c$, $\mathbf{Z}^c$, and vector $\boldsymbol{z}^c$.

$$\max_{\phi} \mathbb{E}_{c\sim\mathcal{D}}\left[\mathbb{E}_{q_\phi(\boldsymbol{Z}|\boldsymbol{X},\boldsymbol{A})}[\log p(\boldsymbol{A}|\boldsymbol{Z})] - \mathrm{KL}[q(\boldsymbol{Z}|\boldsymbol{X},\boldsymbol{A})||p(\boldsymbol{Z})]\right], \tag{5}$$

where inference model $q_\phi(\boldsymbol{Z}|\boldsymbol{X},\boldsymbol{A}) = \prod_{i=1}^{N} q_\phi(\boldsymbol{z}_i|\boldsymbol{X},\boldsymbol{A})$, and generative model $p(\boldsymbol{A}|\boldsymbol{Z}) = \prod_{i=1}^{N}\prod_{j=1}^{N}\sigma(\boldsymbol{z}_i^{\mathrm{T}}\boldsymbol{z}_j)$. In addition, we have $\boldsymbol{Z} = [\boldsymbol{z}_1, \boldsymbol{z}_2, \cdots, \boldsymbol{z}_N]^{\mathrm{T}}$ as the latent variables to represent nodes. $p(\boldsymbol{Z})$ is a Gaussian prior, and $\sigma(x) = 1/(1+e^{-x})$ is the sigmoid function. $\mathrm{KL}[p(\cdot)||q(\cdot)]$ is the Kullback-Leibler divergence between $p(\cdot)$ and $q(\cdot)$.

As we have collected an expert-level offline dataset, we can remove the dependency of the hindsight information $\mathrm{HI}(c, \boldsymbol{\tau})$ on the expert trajectory $\boldsymbol{\tau}$, thus replacing $\mathrm{HI}(c, \boldsymbol{\tau})$ with $\mathrm{HI}(c)$ and approximating $\mathrm{HI}(c)$ through the learned surrogate inference model q_ϕ in Eqn.(5).

Following prior work (Mirhoseini et al., 2021), we compute the mean of all latent embeddings to obtain a representation for the whole graph $\mathcal{G}^c$, *i.e.,* setting the offline placement hindsight information $\mathrm{HI}(c, \boldsymbol{\tau}) = \mathrm{HI}(c) = \frac{1}{N}\sum_{i=1}^{N}\boldsymbol{z}_i^c$, where $\boldsymbol{z}_i^c$ is a row vector of the matrix $\boldsymbol{Z}^c = q_\phi(\boldsymbol{Z}^c|\boldsymbol{X}^c, \boldsymbol{A}^c)$. The corresponding pseudo-code is shown in Appendix Algo. 1.

Network Architecture. We illustrate the architecture of ChiPFormer at the right of Fig.3, which uses GPT (Radford et al., 2018; 2019) as the backbone network. Specifically, there are $(2T+1)$ tokens in one placement trajectory, including T state tokens, T action tokens, and one circuit token to represent the hindsight information $\mathrm{HI}(c)$.

We determine the placement order by the sorting results of the area and the net number of the macros. We represent the state token $\boldsymbol{s}_t$ using a three-channel input, including the position mask, the wire mask, and the view mask, similar to MaskPlace (Lai et al., 2022). These mask representations are discussed in Appendix A.4. To scale the input size, we divide the chip canvas into an 84×84 grid. Thus each input channel can be seen as an image with the resolution 84×84, and we can represent the state as a three-channel input of size $3 \times 84 \times 84$. For the action token $\boldsymbol{a}_t$, we represent it as the two-dimensional (x, y) coordinates of the macro t, which will be placed in the 2D grid. To avoid overlapping between macros, we also remove all infeasible actions by the position mask introduced in Appendix A.4.

Before feeding the state and the action tokens into the GPT backbone, we employ two trainable embedding models to first encode the states and actions separately. Furthermore, we introduce a circuit token to distinguish different circuit topologies. The main reason is that the visual representation in the state token ignores topology information of circuits, which is important for placement, especially in multi-task learning. If two circuits contain the same modules but are connected differently, the optimal placement solution should be different as Appendix A.3. Thus, we encode the circuit topology information into the hindsight information (Eqn.(4)). As the circuit token, $\mathrm{HI}(c)$ is independent of the placement state token and should be considered in all sequence modeling steps. Therefore, it is put at the beginning of the sequential modeling process. Detailed model architecture and hyper-parameter settings are in Appendix A.5, Table 10 and 11.

3.4. Online Finetuning for Unseen Chip

For a new unseen chip placement task (*i.e.,* $\mathcal{M}_{\text{test}} \notin \mathcal{M}_{\text{train}}^{[n]}$), we can finetune the pretrained ChiPFormer model by few-shot online rollouts similar to the online decision transformer (Zheng et al., 2022). To maintain the training stability of the supervised pretraining process, we finetune the policy via a return-weighted regression through online interaction with the unseen circuit c_{test}:

$$\min_{\theta} \mathcal{L}(\pi_\theta) := -\mathbb{E}_{\mathcal{B}(\boldsymbol{\tau})}\left[\omega(\boldsymbol{\tau})\log\pi_\theta(\boldsymbol{a}_t|\boldsymbol{\tau}_t, \mathrm{HI}(c_{\text{test}}))\right],$$

where the expectation is over the replay buffer $\mathcal{B}$ of online rollout trajectories, $\omega(\boldsymbol{\tau}) := \frac{\exp(R(\boldsymbol{\tau})/\alpha)}{\mathbb{E}_{\mathcal{B}(\boldsymbol{\tau})}[\exp(R(\boldsymbol{\tau})/\alpha)]}$ is the return-guided prioritization weight, $R(\boldsymbol{\tau}) := \sum_{t=1}^{T} r(\boldsymbol{s}_t, \boldsymbol{a}_t)$ is the return of a trajectory, and α is the sampling temperature. In the implementation, we also build the online replay buffer $\mathcal{B}$ on a priority queue to keep the trajectories with the highest returns in the search history instead of the FIFO (First-In First-Out) replay buffer in Zheng et al. (2022).

Further, we employ a policy entropy as an intrinsic motivation $\mathcal{H}(\pi_\theta) = \mathbb{E}_{\mathcal{B}(\boldsymbol{\tau})}\left[-\log\pi_\theta(\cdot|\boldsymbol{s}_t, \mathrm{HI}(c_{\text{test}}))\right]$, encouraging the online rollout to yield more exploratory behaviors. Combining the weighted regression and the entropy objective, we obtain the following policy finetuning objective:

$$\min_{\theta} \mathcal{L}(\pi_\theta) + \lambda\max(0, \beta - \mathcal{H}(\pi_\theta)), \tag{6}$$

where λ is the hyper-parameter to represent the weight of the entropy loss, and we introduce the $\max$ operator to keep the value of entropy $\mathcal{H}(\pi_\theta)$ larger than one threshold β, so as to maintain a certain level of exploration.

3.5. Mixed-size Placement

The mixed-size placement aims at placing all modules on the chip canvas, including macros and standard cells. Our work follows similar procedures in Mirhoseini et al. (2021); Cheng & Yan (2021), which employ the optimized-based method for standard cell placement. Different from these methods that fix the positions of macros and subsequently

Table 1: **Mixed-size placement workflow.** In the implementation, we adopt DREAMPlace (Lin et al., 2020) to conduct the optimization-based placement procedure.

Phase	Operation[1]	Description
1	-	Return the macro placement result from the pretrained (or finetuned) ChiPFormer.
2	GP	Fix all macros placed by ChipFormer, and return a coarse layout by optimization method.
3	GP + LG + DP	Set macros and standard cells movable, and return a global optimal placement layout.

[1] Operation in optimization-based placement method (DREAMPlace, Lin et al. (2020)): GP = Global Placement, LG = Legalization, DP = Detailed Placement.

search remaining positions for standard cells, we let the positions of macros (after macro placement) as the initial layout and allow macros still to be movable when performing optimization-based placement as Table 1, thus reaching the optimal solution.

4. Experiment

Benchmarks and Settings. We evaluate ChiPFormer with the recent advanced RL-based methods GraphPlace (Mirhoseini et al., 2021), DeepPR (Cheng & Yan, 2021), MaskPlace (Lai et al., 2022), PRNet (Cheng et al., 2022), the supervised learning-based methods Flora (Liu et al., 2022a) and GraphPlanner (Liu et al., 2022b), the optimization-based methods DREAMPlace (Lin et al., 2020) and RePlAce (Cheng et al., 2018), and the manual design approach (Human). All previous methods are implemented by their default settings. Our experiments are evaluated over 32 circuits from a public placement benchmark (containing *ISPD05* (Nam et al., 2005), *ICCAD04* (Adya et al., 2009), *Ariane RISC-V CPU* (Zaruba & Benini, 2019) with a total of 26 circuits) and a private realistic industrial placement task (containing 6 circuits), which involves considerably more chip circuits than reported in previous works. More details about the experimental benchmark information and hyperparameter settings can be found in Appendix Table 5, 6, and 11.

Offline Data Collection. To collect our multi-task offline placement data, we employ MaskPlace as the proxy method to learn 12 expert-level behavior (data-collecting) policies over 12 circuits (8 from *ISPD05* and 4 from *ICCAD04*) respectively. Then, we collect 500 expert-level placement results for each circuit using these trained data-collecting policies. Because MaskPlace is based on the stochastic policy, we can ensure that the collected 500 behaviors will not collapse to a single solution for each task, thus exhibiting an essential structural diversity in the offline dataset.

Macro Placement Results. First, we investigate whether our offline placement formulation can produce effective macro placement results. In this case, we use part of the collected offline data to learn a ChiPFormer placement policy and then finetune the pretrained policy to the unseen circuits. Specifically, we divide the 12 circuits used for the offline data collection into four circuit groups according to the index in the circuit name. For example, the 1st group includes the circuits *adaptec1*, *bigblue1* and *ibm01*. Then, we use the three grouped offline data to learn a ChiPFormer policy and use the pretrained ChiPFormer as a finetuning initialization for the unseen circuits in the other group. We report the best HPWL of macro placement within the limited rollout times over the unseen circuits in Table 2, where all results (mean and standard deviation) are achieved across 5 seeds. We can find that ChiPFormer(2k) consistently surpasses the other baselines while improving the placement sample efficiency. We can also observe that even with a substantially small number of rollout times (3k in baselines *v.s.* 300 in ChiPFormer), ChiPFormer(300) can achieve better performance in 10 out of 12 tasks. Due to the superior sample efficiency and the strong performance, we use ChiPFormer(300) as the default finetuning setting in the following experiments when it is not explicitly specified.

Further, we report the overlap ratio and the congestion comparison results in Appendix Table 7 and Fig.11. Our ChiPFormer consistently outperforms baselines and exhibits better results in both metrics.

Mixed-Size Placement Results. Next, we compare our ChiPFormer to prior baselines (Human, DREAMPlace, GraphPlace, PRNet, DeepPR, MaskPlace, Flora, and GraphPlanner) on the mixed-size placement task. We provide the visualization in Fig.4 and the comparison results in Table 3 and Appendix Table 8. Benefiting from the offline pretraining, we can observe that our ChiPFormer can produce state-of-the-art placement quality in all mixed-size placements. ChiPFormer can even reduce the HPWL by more than 10% in several chip placement tasks compared to the best results among the baseline methods.

Time Efficiency. Beyond the training sample efficiency explored above, we investigate the training time efficiency. We provide plots of the HPWL metric over runtime in Fig.5. Our offline ChiPFormer finetuning is consistently competitive with previous online methods with more stable performance and less running time, leading to more than $10\times$ speed-up in time efficiency.

Industrial Chip Placement. In Table 4, we also compare ChiPFormer to the human experts in 6 realistic industrial chip design tasks. Compared to the public placement benchmarks, realistic industrial placement involves more complex constraints and design metrics (Bhasker & Chadha, 2009; Wang et al., 2009). For a fair comparison, we use industrial EDA tool to conduct the mixed-size placement workflow after applying our pretrained ChiPFormer model. We provide the comparison results in Table 4, where we can find that

Table 2: **Comparisons of HPWL ($\times 10^5$) for macro placement (lower is better).** All results are based on the performance of unseen circuits. The number after the method name corresponds to the online rollout times, *i.e.,* the number of placement attempts. In particular, ChiPFormer(1) denotes the zero-shot placement performance. The percentage values in pink or cyan indicate the reduction or increase rates of HPWL compared to state-of-the-art baseline results (that have been underlined). By reducing the rollout times by 10 times compared to the best baseline MaskPlace(3k), our ChiPFormer(300) can still produce the minimal wirelength in 10 out of 12 circuits. Further, ChiPFormer(2k) can achieve state-of-the-art results in all circuits when increasing the rollout times to 2k (which is still smaller than the one required by the previous baselines).

circuit	GraphPlace(50k)	DeepPR(3k)	MaskPlace(3k)	ChiPFormer(1)	ChiPFormer(300)	ChiPFormer(2k)
adaptec1	30.01±2.98	19.91±2.13	7.62±0.67	8.87±0.98	7.02±0.11 (-7.87%)	**6.62±0.05** (-13.12%)
adaptec2	351.71±38.20	203.51±6.27	75.16±4.97	122.37±22.61	70.42±2.67 (-6.30%)	**67.10±5.46** (-10.72%)
adaptec3	358.18±13.95	347.16±4.32	100.24±13.54	107.11±8.84	78.32±2.03 (-21.87%)	**76.70±1.15** (-23.48%)
adaptec4	151.42±9.72	311.86±56.74	87.99±3.25	85.63±7.52	69.42±0.54 (-21.10%)	**68.80±1.59** (-21.81%)
bigblue1	10.58±1.29	23.33±3.65	3.04±0.06	3.11±0.03	2.96±0.04 (-2.63%)	**2.95±0.04** (-2.96%)
bigblue2	14.78±0.95	11.38±0.20	5.75±0.11	6.85±0.26	6.02±0.09 (+4.70%)	**5.44±0.10** (-5.39%)
bigblue3	357.48±47.83	430.48±12.18	90.04±4.83	131.78±17.36	81.48±4.83 (-9.51%)	**72.92±2.56** (-19.01%)
bigblue4	440.70±15.95	433.90±5.26	103.26±2.69	136.79±13.93	110.10±0.23 (+6.62%)	**102.84±0.15** (-0.41%)
ibm01	4.12±0.10	6.62±0.27	3.73±0.12	4.57±0.27	3.61±0.08 (-3.22%)	**3.05±0.11** (-18.23%)
ibm02	4.56±0.03	6.33±0.05	4.85±0.34	6.01±0.41	4.84±0.17 (-0.21%)	**4.24±0.25** (-12.58%)
ibm03	2.57±0.07	3.08±0.17	1.82±0.10	2.15±0.17	1.75±0.07 (-3.85%)	**1.64±0.06** (-9.89%)
ibm04	5.73±0.15	4.80±0.26	4.73±0.07	5.00±0.14	4.19±0.11 (-11.42%)	**4.06±0.13** (-14.16%)

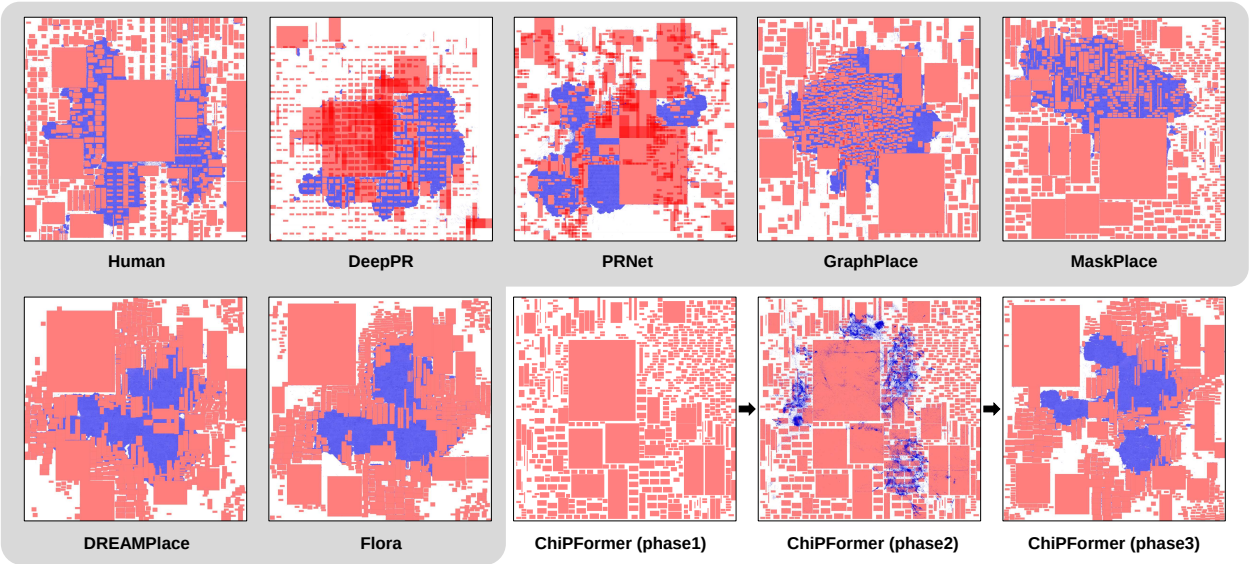


Figure 4: **Visualization of mixed-size placement for circuit *adaptec3*.** Red marks macros and blue marks standard cells. Baseline methods (shaded background) include Human (HPWL = 19.41×10^7, Overlap = 0%), DeepPR (HPWL = 24.11×10^7, Overlap = 24.35%), PRNet (HPWL = 23.24×10^7, Overlap = 10.71%), GraphPlace (HPWL = 25.80×10^7, Overlap = 1.24%), MaskPlace (HPWL = 21.49×10^7, Overlap = 0%), DREAMPlace (HPWL = 15.63×10^7, Overlap = 0%), Flora (HPWL = 15.65×10^7, Overlap = 0%), and ChiPFormer (HPWL = $\underline{13.97\times10^7}$, Overlap = $\underline{0\%}$). ChiPFormer (phase 1), (phase 2), and (phase 3) represent the return results of phase 1, 2, and 3 in our placement flow (Table 1), respectively.

ChiPFormer can exceed expert human performance in most industrial metrics.

Dataset Size. To explore the impact of training dataset size on the transfer ability of our ChiPFormer, we construct three offline datasets (small, medium, and large) containing 1, 3, and 9 circuits, respectively. For each dataset, we compare the HPWL results of ChiPFormer(1), ChiPFormer(300), and ChiPFormer(2k) after finetuning for circuit *adaptec1* over 10 seeds in Fig.6 (a). The results show that using more offline circuits, ChiPFormer can yield better generalization for new unseen circuits, especially in the zero-shot transfer settings (ChiPFormer(1)).

Table 3: **Comparisons of HPWL ($\times 10^7$) for the mixed-size placement (lower is better).** The baseline results for DREAMPlace, GraphPlace, PRNet, DeepPR, MaskPlace, Flora, and GraphPlanner are taken from the respective papers. Baseline *Human* means the macros are placed by hardware experts. We implement our ChiPFormer method according to the workflow specified in Table 1. The percentage value in the pink denotes the reduction rate of HPWL compared to the best results (underlined) among baselines.

circuit	Human	DREAMPlace	GraphPlace	PRNet	DeepPR	MaskPlace	Flora	GraphPlanner	ChiPFormer(workflow)
adaptec1	7.33	6.56	8.67	8.28	8.01	7.93	6.47	6.55	**6.45±0.02** (-0.31%)
adaptec2	8.22	10.11	12.41	12.33	12.32	9.95	7.77	7.75	**7.36±0.26** (-5.03%)
adaptec3	19.41	15.63	25.80	23.24	24.11	21.49	15.65	15.08	**13.97±0.80** (-7.36%)
adaptec4	17.44	14.41	25.58	23.40	23.64	22.97	14.30	14.27	**12.97±0.29** (-9.30%)
bigblue1	8.94	8.52	16.85	14.10	14.04	9.43	8.51	8.59	**8.48±0.02** (-0.35%)
bigblue2	13.67	12.57	14.20	14.48	14.04	14.13	12.59	12.72	**9.86±0.32** (-21.56%)
bigblue3	30.40	46.06	36.48	46.86	45.06	37.29	-	-	**27.33±0.31** (-10.10%)
bigblue4	74.38	79.50	104.00	100.13	95.20	106.18	74.76	-	**65.98±4.08** (-11.29%)

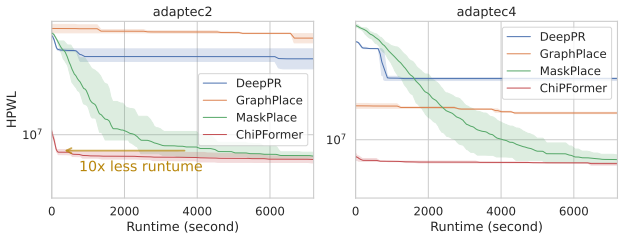
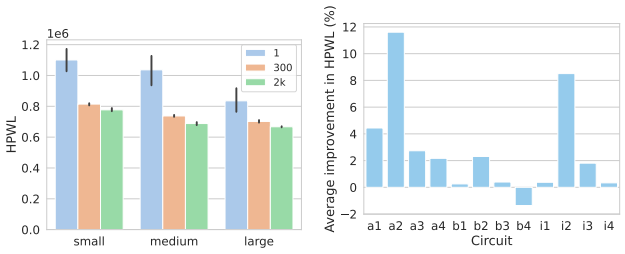


Figure 5: **HPWL curves over runtime.** All methods are evaluated on two RTX3090 GPUs. ChiPFormer can search for high-quality placements in a very short time, more than 10 times faster compared with other methods.



(a) (b)

Figure 6: **(a) Comparison of different offline dataset sizes.** Small, medium and large datasets include 1/3/9 offline circuits, respectively. We can see that the larger dataset tends to generate higher-quality placements. **(b) Performance improvement when introducing the circuit token.** We can observe that introducing the circuit token contributes to improved performance in most circuits.

Table 4: **Comparisons to human expert design on 6 industrial chip design tasks.** The lower the metrics the better.

circuit	method	Timing		NVP[1]	Congestion	
		WNS/ps	TNS/ns		H/%	V/%
C1	Human	204	57.1	2569	0.06	0.38
	MaskPlace	161	42.7	1964	0.07	**0.07**
	ChiPFormer	**142**	**19.4**	**1636**	**0.04**	**0.07**
C2	Human	403	492.2	11360	0.63	2.05
	MaskPlace	242	259.1	9710	0.57	1.67
	ChiPFormer	**177**	**224.9**	**8110**	**0.53**	**1.27**
C3	Human	**102**	91.9	5614	**1.02**	0.85
	MaskPlace	116	92.8	5559	1.05	0.87
	ChiPFormer	108	**91.2**	**5452**	**1.02**	**0.82**
C4	Human	399	438.0	13925	0.97	**0.34**
	MaskPlace	389	324.2	12582	0.68	**0.34**
	ChiPFormer	**248**	**266.0**	**12398**	**0.62**	**0.34**
C5	Human	89	10.8	2675	**0.02**	0.07
	MaskPlace	122	32.2	2975	**0.02**	0.22
	ChiPFormer	**80**	**4.9**	**1706**	**0.02**	**0.04**
C6	Human	154	137.4	6833	0.70	0.22
	MaskPlace	81	49.6	7040	0.77	0.26
	ChiPFormer	**78**	**38.1**	**6412**	**0.63**	**0.22**

[1] NVP = Number of Violation Points, WNS = Worst Negative Slack, TNS = Total Negative Slack. More details about metrics can be found in Bhasker & Chadha (2009) and Wang et al. (2009).

Ablation Study. To study the role of the multi-task hindsight information matching and verify the effects of circuit token HI(c), we ablate the circuit token and keep other settings the same. We test the zero-shot performance on 12 circuits in benchmark *ISPD05* and *ICCAD04*. In Fig.6 (b), we measure the difference between the ablated ChiPFormer and the full ChiPFormer. We can find that when introducing the circuit token, ChiPFormer can acquire a better generalization ability to zero-shot to unseen tasks.

Further, we compare FIFO and priority-based storage strategies for the finetuning replay buffer and ablate the entropy loss in Eqn.(6). We show the ablation results in Fig.7. We can see that the FIFO-based strategy tends to forget the experiences with higher returns and lead to sub-optimal and high-variance placement results. When removing the entropy loss, ChiPFormer suffers from a lack of exploration ability and converges to a local optimum.

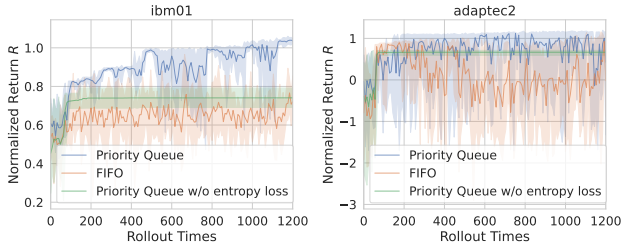


Figure 7: **Ablation results in replay buffer update rules and the entropy motivation in ChiPFormer finetuning.**

5. Conclusion

In this paper, we propose ChiPFormer, an offline RL method for chip placement tasks, and design a mixed-size placement workflow. ChiPFormer can learn from existing placement solutions (offline data) and significantly improve the training sample and time efficiency. ChiPFormer can also acquire strong zero-shot transfer ability and yield an effective initialization for the few-shot finetuning on unseen chip placement tasks.

Acknowledgements

This paper is partially supported by the National Key R&D Program of China No.2022ZD0161000 and the General Research Fund of Hong Kong No.17200622.

References

Adya, S., Chaturvedi, S., and Markov, I. Iccad'04 mixed-size placement benchmarks. *GSRC Bookshelf*, 2009.

Bhasker, J. and Chadha, R. *Static timing analysis for nanometer designs: A practical approach*. Springer Science & Business Media, 2009.

Brenner, U., Hermann, A., Hoppmann, N., and Ochsendorf, P. Bonnplace: A self-stabilizing placement framework. In *Proceedings of the 2015 Symposium on International Symposium on Physical Design (ISPD)*, pp. 9–16, 2015.

Chan, T. F., Cong, J., Shinnerl, J. R., Sze, K., and Xie, M. mpl6: Enhanced multilevel mixed-size placement. In *Proceedings of the 2006 international symposium on Physical design (ISPD)*, pp. 212–214, 2006.

Chen, L., Lu, K., Rajeswaran, A., Lee, K., Grover, A., Laskin, M., Abbeel, P., Srinivas, A., and Mordatch, I. Decision transformer: Reinforcement learning via sequence modeling. *Advances in neural information processing systems (NeurIPS)*, 34:15084–15097, 2021.

Chen, T.-C., Jiang, Z.-W., Hsu, T.-C., Chen, H.-C., and Chang, Y.-W. Ntuplace3: An analytical placer for large-scale mixed-size designs with preplaced blocks and density constraints. *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems (TCAD)*, 27(7): 1228–1240, 2008.

Cheng, C.-K., Kahng, A. B., Kang, I., and Wang, L. Replace: Advancing solution quality and routability validation in global placement. *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems (TCAD)*, 38 (9):1717–1730, 2018.

Cheng, R. and Yan, J. On joint learning for solving placement and routing in chip design. *Advances in Neural Information Processing Systems (NeurIPS)*, 34, 2021.

Cheng, R., Lyu, X., Li, Y., Ye, J., Jianye, H., and Yan, J. The policy-gradient placement and generative routing neural networks for chip design. In *Advances in Neural Information Processing Systems (NeurIPS)*, 2022.

Emmons, S., Eysenbach, B., Kostrikov, I., and Levine, S. Rvs: What is essential for offline rl via supervised learning? *arXiv preprint arXiv:2112.10751*, 2021.

Furuta, H., Matsuo, Y., and Gu, S. S. Generalized decision transformer for offline hindsight information matching. *arXiv preprint arXiv:2111.10364*, 2021.

Garey, M. R. and Johnson, D. S. The rectilinear steiner tree problem is np-complete. *SIAM Journal on Applied Mathematics*, 32(4):826–834, 1977.

Ghosh, D., Gupta, A., Reddy, A., Fu, J., Devin, C., Eysenbach, B., and Levine, S. Learning to reach goals via iterated supervised learning. *arXiv preprint arXiv:1912.06088*, 2019.

Gu, J., Jiang, Z., Lin, Y., and Pan, D. Z. Dreamplace 3.0: Multi-electrostatics based robust vlsi placement with region constraints. In *2020 IEEE/ACM International Conference On Computer Aided Design (ICCAD)*, pp. 1–9. IEEE, 2020.

Jiang, Z., Songhori, E., Wang, S., Goldie, A., Mirhoseini, A., Jiang, J., Lee, Y.-J., and Pan, D. Z. Delving into macro placement with reinforcement learning. In *2021 ACM/IEEE 3rd Workshop on Machine Learning for CAD (MLCAD)*, pp. 1–3. IEEE, 2021.

Kahng, A. B., Reda, S., and Wang, Q. Aplace: A general analytic placement framework. In *Proceedings of the 2005 international symposium on Physical design (ISPD)*, pp. 233–235, 2005.

Khatkhate, A., Li, C., Agnihotri, A. R., Yildiz, M. C., Ono, S., Koh, C.-K., and Madden, P. H. Recursive bisection based mixed block placement. In *Proceedings of the 2004 international symposium on Physical design (ISPD)*, pp. 84–89, 2004.

Kim, H., Kim, M., Kim, J., and Park, J. Collaborative symmetricity exploitation for offline learning of hardware design solver. In *3rd Offline RL Workshop: Offline RL as a"Launchpad"*.

Kim, M.-C. and Markov, I. L. Complx: A competitive primal-dual lagrange optimization for global placement. In *Proceedings of the 49th Annual Design Automation Conference (DAC)*, pp. 747–752, 2012.

Kim, M.-C., Viswanathan, N., Alpert, C. J., Markov, I. L., and Ramji, S. Maple: Multilevel adaptive placement for mixed-size designs. In *Proceedings of the 2012 ACM international symposium on International Symposium on Physical Design (ISPD)*, pp. 193–200, 2012.

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

Kipf, T. N. and Welling, M. Variational graph auto-encoders. *arXiv preprint arXiv:1611.07308*, 2016b.

Lai, Y., Mu, Y., and Luo, P. Maskplace: Fast chip placement via reinforced visual representation learning. In *Advances in Neural Information Processing Systems (NeurIPS)*, 2022.

Lin, T., Chu, C., Shinnerl, J. R., Bustany, I., and Nedelchev, I. Polar: Placement based on novel rough legalization and refinement. In *2013 IEEE/ACM International Conference on Computer-Aided Design (ICCAD)*, pp. 357–362. IEEE, 2013.

Lin, Y., Jiang, Z., Gu, J., Li, W., Dhar, S., Ren, H., Khailany, B., and Pan, D. Z. Dreamplace: Deep learning toolkit-enabled gpu acceleration for modern vlsi placement. *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems (TCAD)*, 40(4):748–761, 2020.

Liu, Y., Ju, Z., Li, Z., Dong, M., Zhou, H., Wang, J., Yang, F., Zeng, X., and Shang, L. Floorplanning with graph attention. In *Proceedings of the 59th ACM/IEEE Design Automation Conference (DAC)*, pp. 1303–1308, 2022a.

Liu, Y., Ju, Z., Li, Z., Dong, M., Zhou, H., Wang, J., Yang, F., Zeng, X., and Shang, L. Graphplanner: Floorplanning with graph neural network. *ACM Transactions on Design Automation of Electronic Systems (TODAES)*, 2022b.

Lu, J., Chen, P., Chang, C.-C., Sha, L., Dennis, J., Huang, H., Teng, C.-C., and Cheng, C.-K. eplace: Electrostatics based placement using nesterov's method. In *2014 51st ACM/EDAC/IEEE Design Automation Conference (DAC)*, pp. 1–6. IEEE, 2014.

Lu, Y.-C., Yang, T., Lim, S. K., and Ren, H. Placement optimization via ppa-directed graph clustering. In *2022 ACM/IEEE 4th Workshop on Machine Learning for CAD (MLCAD)*, pp. 1–6. IEEE, 2022.

Mirhoseini, A., Goldie, A., Yazgan, M., Jiang, J. W., Songhori, E., Wang, S., Lee, Y.-J., Johnson, E., Pathak, O., Nazi, A., et al. A graph placement methodology for fast chip design. *Nature*, 594(7862):207–212, 2021.

Nam, G.-J., Alpert, C. J., Villarrubia, P., Winter, B., and Yildiz, M. The ispd2005 placement contest and benchmark suite. In *Proceedings of the 2005 international symposium on Physical design (ISPD)*, pp. 216–220, 2005.

Nocedal, J. and Wright, S. J. Quadratic programming. *Numerical optimization*, pp. 448–492, 2006.

Radford, A., Narasimhan, K., Salimans, T., Sutskever, I., et al. Improving language understanding by generative pre-training. 2018.

Radford, A., Wu, J., Child, R., Luan, D., Amodei, D., Sutskever, I., et al. Language models are unsupervised multitask learners. *OpenAI blog*, 1(8):9, 2019.

Reed, S., Zolna, K., Parisotto, E., Colmenarejo, S. G., Novikov, A., Barth-Maron, G., Gimenez, M., Sulsky, Y., Kay, J., Springenberg, J. T., et al. s. *arXiv preprint arXiv:2205.06175*, 2022.

Roy, J. A., Adya, S. N., Papa, D. A., and Markov, I. L. Min-cut floorplacement. *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems (TCAD)*, 25(7):1313–1326, 2006.

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

Spindler, P., Schlichtmann, U., and Johannes, F. M. Kraftwerk2—a fast force-directed quadratic placement approach using an accurate net model. *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems (TCAD)*, 27(8):1398–1411, 2008.

Sutton, R. S., McAllester, D., Singh, S., and Mansour, Y. Policy gradient methods for reinforcement learning with function approximation. *Advances in neural information processing systems (NeurIPS)*, 12, 1999.

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

Veličković, P., Cucurull, G., Casanova, A., Romero, A., Lio, P., and Bengio, Y. Graph attention networks. *arXiv preprint arXiv:1710.10903*, 2017.

Viswanathan, N., Nam, G.-J., Alpert, C. J., Villarrubia, P., Ren, H., and Chu, C. Rql: Global placement via relaxed quadratic spreading and linearization. In *Proceedings of the 44th annual Design Automation Conference (DAC)*, pp. 453–458, 2007a.

Viswanathan, N., Pan, M., and Chu, C. Fastplace 3.0: A fast multilevel quadratic placement algorithm with placement congestion control. In *2007 Asia and South Pacific Design Automation Conference (ASP-DAC)*, pp. 135–140. IEEE, 2007b.

Wang, L.-T., Chang, Y.-W., and Cheng, K.-T. T. *Electronic design automation: synthesis, verification, and test*. Morgan Kaufmann, 2009.

Yang, X., Sarrafzadeh, M., et al. Dragon2000: Standard-cell placement tool for large industry circuits. In *IEEE/ACM International Conference on Computer Aided Design. (ICCAD)*, pp. 260–263. IEEE, 2000.

Zaruba, F. and Benini, L. The cost of application-class processing: Energy and performance analysis of a linux-ready 1.7-ghz 64-bit risc-v core in 22-nm fdsoi technology. *IEEE Transactions on Very Large Scale Integration (VLSI) Systems (TVLSI)*, 27(11):2629–2640, 2019.

Zheng, Q., Zhang, A., and Grover, A. Online decision transformer. *arXiv preprint arXiv:2202.05607*, 2022.

A. Appendix

A.1. Statistics of Benchmark Circuits

In Table 5 and 6, we provide the statistics of 26 circuits from benchmarks *ISPD05*, *ICCAD04*, and *Ariane RISC-V CPU* and 6 realistic industrial circuits used in our experiments, where Macro (placed 1st) means (the number of) macros placed by decision transformer. For the three circuits *adaptec1*, *adaptec2*, and *bigblue1*, because they have fixed macros that have been pre-placed around the chip as IOs, we maintain the positions of them but still compute HPWL with respect to these fixed macros to make a fair comparison. For circuits *bigblue2* and *bigblue4* that contain more than 8k macros, we select part of the macros to place based on the importance ordering. For circuits in benchmark *ICCAD04*, the circuit description files do not distinguish between macros and standard cells. Thus we select 256 modules as macros and the remaining modules as standard cells. In the implementation, we use the same circuit settings in all experiments.

Table 5: Statistics of public benchmark circuits.

Circuit	Macros	Macros (placed 1st)	Hard Macros	Standard Cells	Nets	Pins	Ports	Area Util(%)
adaptec1	543	63	63	210904	221142	944063	0	55.62
adaptec2	566	159	159	254457	266009	1069482	0	74.46
adaptec3	723	723	201	450927	466758	1875039	0	61.51
adaptec4	1329	1329	92	494716	515951	1912420	0	48.62
bigblue1	560	32	32	277604	284479	1144691	0	31.58
bigblue2	23084	256	52	534782	577235	2122282	0	32.43
bigblue3	1293	1293	138	1095519	1123170	3833218	0	66.81
bigblue4	8170	1024	52	2169183	2229886	8900078	0	35.68
ariane	932	932	134	0	12404	22802	1231	78.39
ibm01	256	256	52	12506	14111	50566	246	61.94
ibm02	256	256	52	19321	19584	81199	259	64.63
ibm03	256	256	52	22846	27401	93573	283	57.97
ibm04	256	256	52	26899	31970	105859	287	54.88
ibm06	256	256	52	32320	34826	128182	166	54.77
ibm07	256	256	52	45419	48117	175639	287	46.03
ibm08	256	256	52	51000	50513	204890	286	47.13
ibm09	256	256	52	53142	60902	222088	285	44.52
ibm10	256	256	52	68643	75196	297567	744	61.40
ibm11	256	256	52	70185	81454	280786	406	41.40
ibm12	256	256	52	70425	77240	317760	637	53.85
ibm13	256	256	52	83775	99666	357075	490	39.43
ibm14	256	256	52	146991	152772	546816	517	22.49
ibm15	256	256	52	161177	186608	715823	383	28.89
ibm16	256	256	52	183026	190048	778823	504	39.46
ibm17	256	256	52	184735	189581	860036	743	19.11
ibm18	256	256	52	210328	201920	819697	272	11.09

Table 6: Statistics of industrial benchmark circuits.

Circuit	Macros	Macros (placed 1st)	Standard Cells	Nets	Pins	Ports
C1	45	45	652519	817469	2792845	9009
C2	159	159	3487686	4417159	9532116	6676
C3	134	134	2268372	2776101	10004114	1416
C4	70	70	1058352	1340788	4256481	27120
C5	144	144	2766094	3143780	11519226	11174
C6	165	165	1777410	2443917	7268601	14882

A.2. Additional Results

In Fig.8, we provide more visualization results for mixed-size placement. In Fig.9, we provide additional results (on circuit *adaptec4* and *bigblue2*) for the training sample efficiency comparison. In Fig.10, we present additional results (over benchmarks *ISPD05* and *ICCAD04*) for the training time efficiency comparison. In Table 7, we provide the comparison results of the overlap ratio for macro placement. In Table 8, we give additional results for the mixed-size placement in

ICCAD04 benchmark. In Table 9, we give out the reconstruct quality of testing set and the related code is from original repository [2].

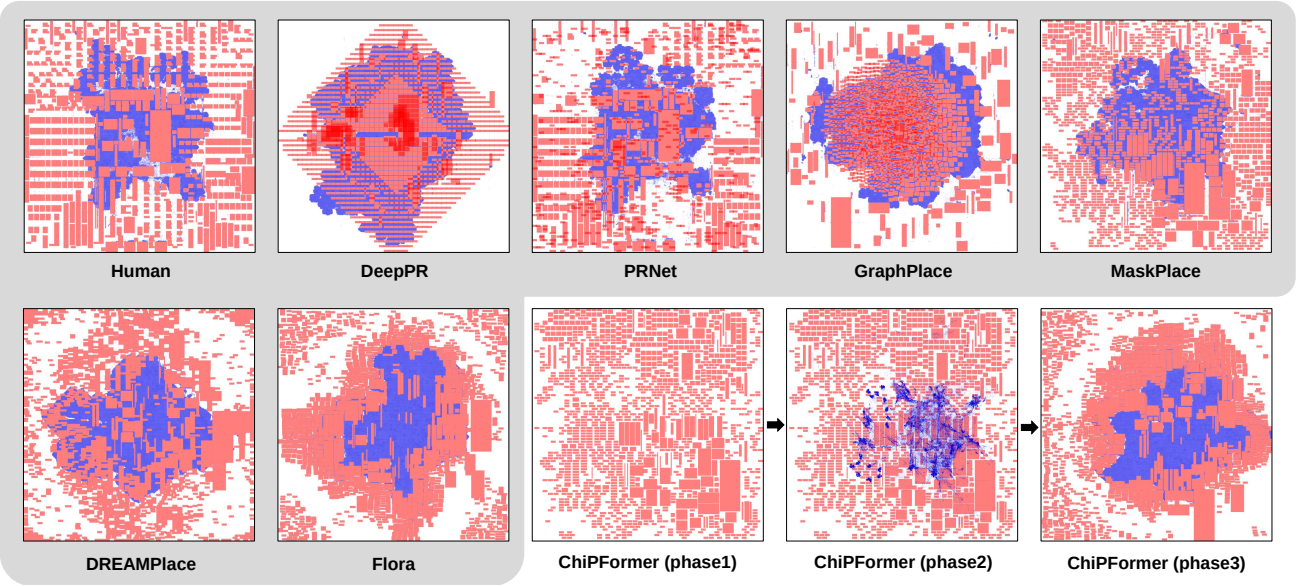


Figure 8: **Visualization of mixed-size placement for circuit *adaptec4*.** Results include Human (HPWL = 17.44×10^7, Overlap = 0%), DeepPR (HPWL = 23.40×10^7, Overlap = 32.11%), PRNet (HPWL = 23.64×10^7, Overlap = 13.36%), GraphPlace (HPWL = 25.58×10^7, Overlap = 7.43%), MaskPlace (HPWL = 22.97×10^7, Overlap = 0%), DREAMPlace (HPWL = 14.41×10^7, Overlap = 0%), Flora (HPWL = 14.30×10^7, Overlap = 0%), and ChiPFormer (HPWL = $\underline{12.97\times10^7}$, Overlap = $\underline{0\%}$).

Congestion Constraint. Considering that some scenarios require the congestion metric, we test ChiPFormer on both HPWL and congestion metrics. We run ChiPFormer on 8 circuits in *ISPD05*. To make a fair comparison, we run experiments on the processed circuits from PRNet (Cheng et al., 2022). We show the results in Fig.11. At testing, we first set the congestion threshold to $+\infty$ and find how far congestion can go when considering only HPWL. Then, we gradually reduce this threshold to find all points on the Pareto front. We can find that our ChiPFormer can achieve better results in both HPWL and congestion metrics when compared to MaskPlace, DeepPR and PRNet.

A.3. Example on the Circuit Token

The agent learns policy only from state and action tokens when ablating the circuit token (*i.e.,* ignoring the topology structure of the circuit). Suppose two circuits contain the same modules but are connected differently. In that case, the agent can not

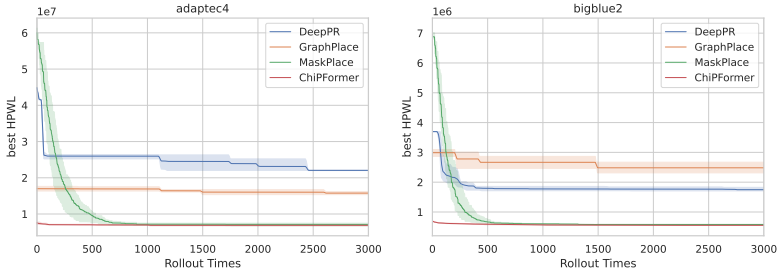


Figure 9: **HPWL versus rollout times (number of trajectories).** ChiPformer can reduce the rollout times by more than 90% while achieving the same placement quality.

[2] https://github.com/DaehanKim/vgae_pytorch

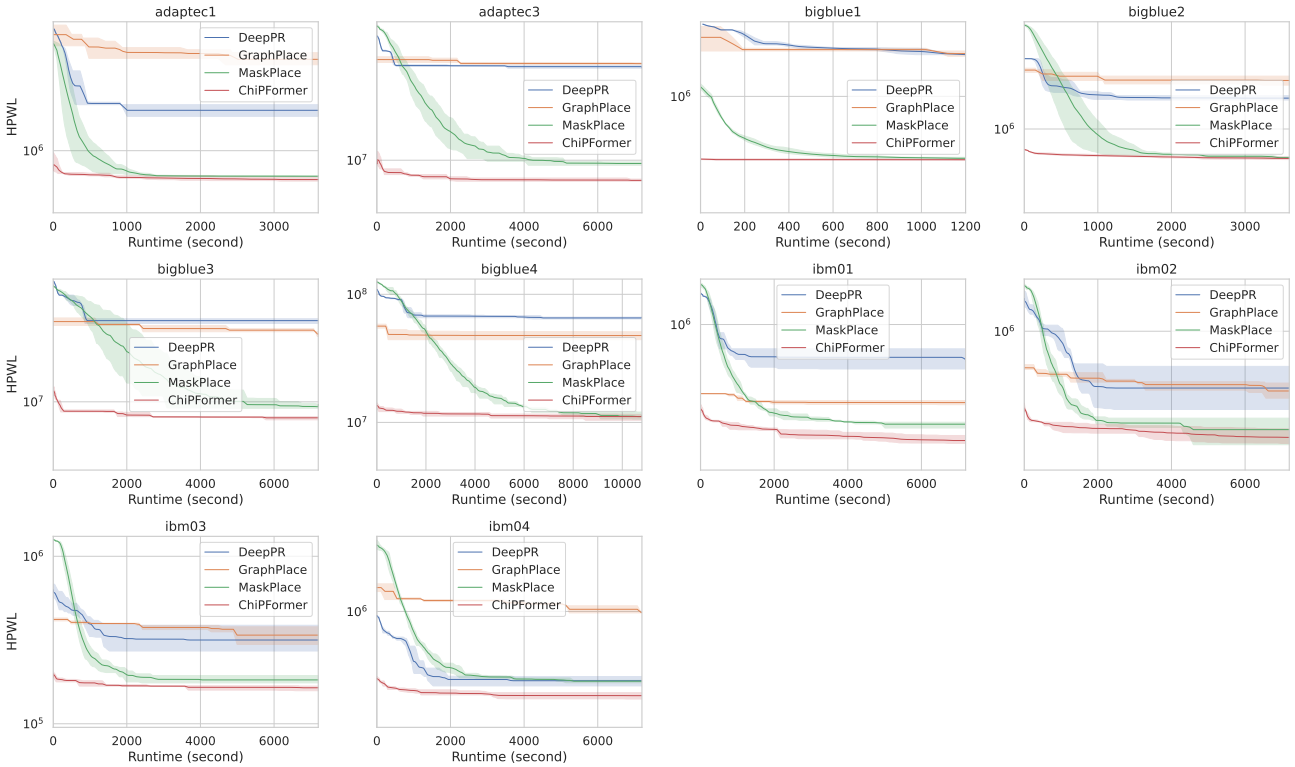


Figure 10: **HPWL curves over runtime. (cont.)** All methods are evaluated on two RTX3090 GPUs.

Table 7: **Comparisons of the overlap ratio (%) for macro placement**, which are evaluated by the ratio of the cumulative overlapping area to the total area in the chip. The overlap ratio should be 0 (or as close to 0 as possible) to meet the requirements of chip manufacturing. We can observe that ChipFormer can maintain non-overlapping placements in most circuits.

circuit	GraphPlace	DeepPR	PRNet	MaskPlace	ChiPFormer
adaptec1	1.89	46.26	12.60	0.00	0.00
adaptec2	1.54	18.39	9.94	0.00	0.00
adaptec3	1.24	24.35	10.71	0.00	0.00
adaptec4	7.59	32.11	13.36	0.00	0.00
bigblue1	1.98	2.08	2.04	0.00	0.00
bigblue2	0.77	29.42	20.06	0.00	0.00
bigblue3	0.96	74.03	5.11	0.00	0.00
bigblue4	5.54	8.35	6.46	0.00	0.00
ariane	5.13	38.91	9.49	3.33	3.27

distinguish them because they share the same state and action tokens and thus generate the same actions in the two circuits. However, that will lead to a sub-optimal solution on one circuit. To illustrate, we give an example in Fig.12.

A.4. State Token Generation

The state tokens in ChiPFormer consist of the view mask, position mask, and wire mask as in MaskPlace (Lai et al., 2022), which are representations of the placement state from different perspectives. Each mask occupies one input channel. All masks can be seen as images with resolution $N \times N$. A state token generation example can be seen in Fig.13. In this case, there are already two placed macros, and now the agent will take action to place Macro 3. There are two nets, and

Table 8: **Comparisons of HPWL ($\times 10^7$) for mixed-size placement in *ICCAD04* benchmark.** HPWL is the smaller the better. SA (simulated annealing) method is implemented as in Mirhoseini et al. (2021). ChiPFormer(workflow) can achieve the best placement quality.

circuit	SA	RePlAce	GraphPlace	MaskPlace	ChiPFormer(workflow)
ibm01	25.85	22.82	31.71	24.18	**16.70** (-26.82%)
ibm02	54.87	47.59	55.11	47.45	**37.87** (-20.19%)
ibm03	80.68	64.36	80.00	71.37	**57.63** (-10.46%)
ibm04	83.32	72.61	86.86	78.76	**65.27** (-10.11%)
ibm06	69.09	58.07	63.48	55.70	**52.57** (-5.62%)
ibm07	111.03	98.57	117.70	95.27	**86.20** (-9.52%)
ibm08	131.07	114.67	134.77	120.64	**102.26** (-10.82%)
ibm09	135.45	120.01	148.74	122.91	**105.61** (-12.00%)
ibm10	423.14	274.29	440.78	367.55	**230.39** (-16.00%)
ibm11	210.12	169.98	218.73	202.23	**160.60** (-5.52%)
ibm12	410.05	306.33	438.57	397.25	**273.14** (-10.83%)
ibm13	259.89	220.14	278.92	246.49	**197.20** (-10.42%)
ibm14	405.80	341.80	455.32	302.67	**301.28** (-0.46%)
ibm15	510.06	451.36	520.06	457.86	**429.71** (-4.80%)
ibm16	614.54	516.05	642.08	584.67	**463.32** (-10.22%)
ibm17	720.40	635.93	814.37	643.75	**569.13** (-10.50%)
ibm18	442.00	399.43	450.67	398.83	**370.36** (-7.14%)

Table 9: **Reconstruct quality of VGAE.** AUC, AP, and ACC mean area under the receiver operating characteristic curve, average precision from prediction scores, and accuracy, respectively.

Test Metrics	AUC	AP	ACC
Value	0.9600	0.9557	0.8969

the corresponding pins and bounding boxes are marked in red and green, respectively. (1) View mask marks all occupied grid cells by macros with 1. (2) Position mask labels all feasible cells that will not generate an overlap if the next macro is placed at the position (supposed that the placement position corresponds to the lower-left corner of the macro). (3) Wire mask marks the HPWL increase when the next macro is placed in the corresponding cells. According to the definition of HPWL, the increase in HPWL is the increase in the size of the net bounding box. Thus, if the place position is inside the net bounding box, the HPWL increase equals 0. Otherwise, it equals the distance from the cells to the bounding box. It is easy to find that the values in the wire mask are the sum of the distances from the cells to all net bounding boxes as Fig.13. Considering that some pins are not located in the lower-left corner of the macro (an offset), the corresponding net bounding box needs to be moved by an equivalent offset (*i.e.,* the bounding box is moved in the opposite direction of the offset vector).

A.5. Model Architecture and Hyper-parameters

The detailed model architecture can be seen in Table 10. Hyper-parameters of the ChiPFormer can be found in Table 11. When pretraining, we keep the sequence length T=256. If a circuit contains more than 256 macros, the trajectory is truncated to the beginning of 256 macros according to the placement order. On the contrary, if there are less than 256 macros, we pad the trajectory with zeros to the fixed length. When testing, for those circuits with more than $T = 256$ macros, ChiPFormer keeps a sliding window to record the latest 256 state tokens and 256 action tokens to generate the next action, following the way of the decision transformer.

A.6. Pseudo-code for Circuit Token Generation

The pseudo-code version for generating circuit token HI(c_{test}) of circuit c_{test} can be seen in Algo. 1.

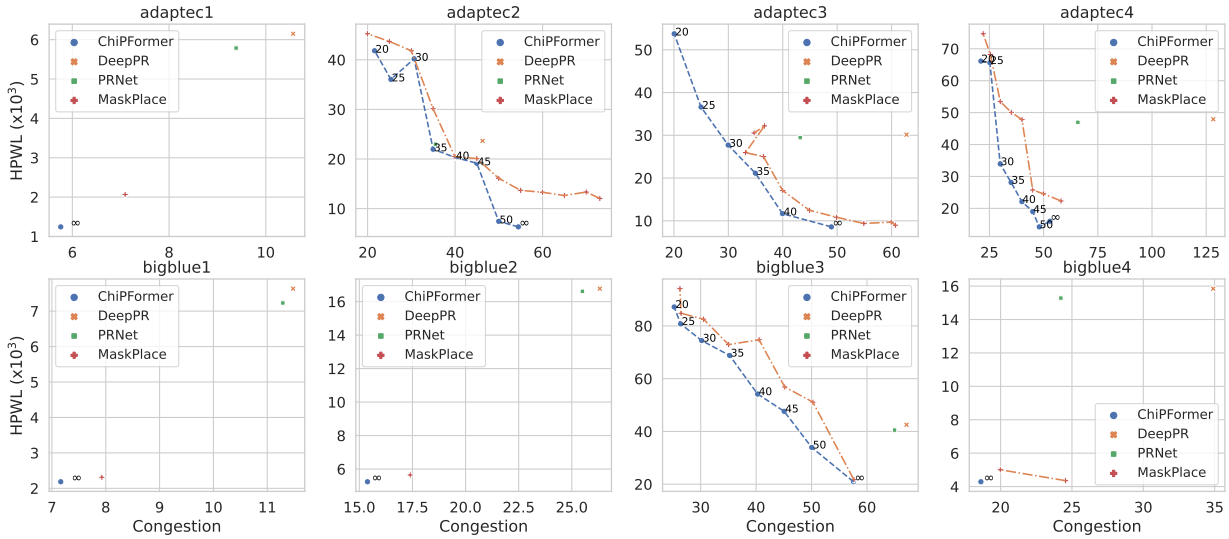


Figure 11: **Comparisons over HPWL and congestion.** Both HPWL and congestion are smaller the better. Numbers beside points are the expected congestion thresholds C (as the input parameter). We run ChiPFormer finetuning with 300 rollouts, then set the congestion threshold to $+\infty$ and gradually decrease it.

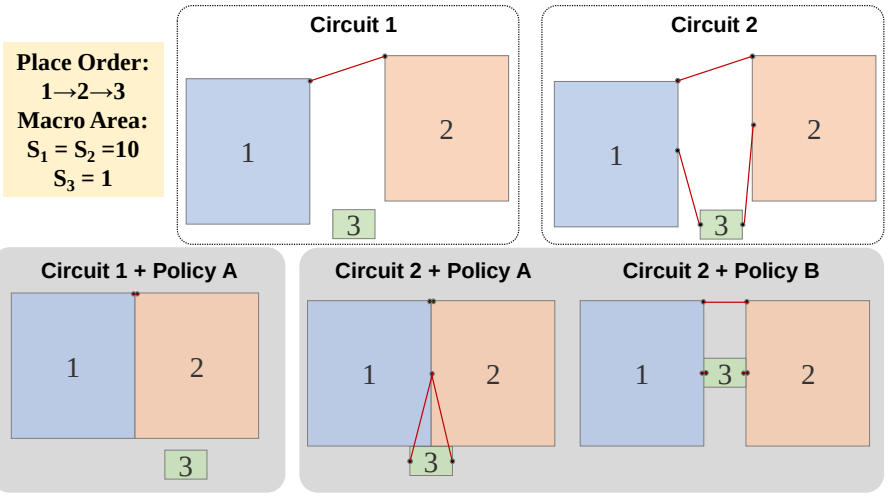


Figure 12: **An example for two circuits that share the same state and action tokens (s_1, a_1, s_2, a_2).** Consider the two circuits (circuit 1 and circuit 2). The only difference between them is that the 3rd macro of the circuit is connected to both the 1st and 2nd macros in circuit 2. According to the definition of state token, the state tokens s_1, s_2 are the same because the position, wire, and view masks only involve the placed macros and the current macro to be placed. That means state s_1 contains the 1st macro and state s_2 contains the 1st and 2nd macros, which are the same in two circuits. Consider an optimal policy A that minimizes the HPWL in circuit 1 (subplot "Circuit 1 + Policy A"). However, deploying policy A over circuit 2 will lead to the sub-optimal solution (subplot "Circuit 2 + Policy A"). In contrast, the optimal policy for circuit 2 should be Policy B, which renders different behaviors at state s_2 (subplot "Circuit 2 + Policy B"). Thus, in the multi-task chip placement setting, the circuit token helps us identify optimal placement behaviors when circuits contain the same macros while differing in their topology structures.

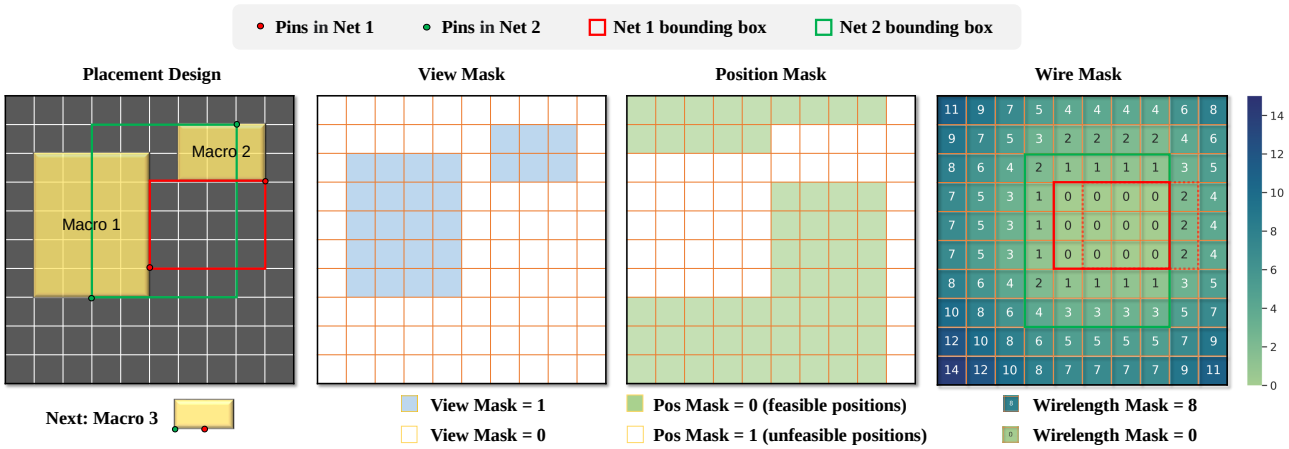


Figure 13: **An example for state tokens (placement masks) generation.** The view mask, position mask, and wire mask are generated based on the current placement design and the next macro to place. View mask $V \in \{0,1\}^{N\times N}$ is to mark the occupied cells by macros. Position mask $P \in \{0,1\}^{N\times N}$ is to label all feasible action positions that do not cause overlap. Wire mask $W \in [0,1]^{N\times N}$is to label the HPWL increase when the next macro is placed in the corresponding position. For ease of understanding, we marked the data before normalization for the wire mask, which is the sum of the shortest Manhattan distances for each cell to the inside of bounding boxes.

Table 10: Model Architecture

	layer name (index)	kernel size	output size
circuit token extraction (VGAE)	GCN_1	-	(num of macros, 32)
	GCN_2	-	(num of macros, 32)
	Pooling	-	(32,)
circuit embedding	FC_1	-	(1024,)
	FC_2	-	(1024,)
	FC_3	-	(768,)
state embedding	CNN_1	8×8, 16	(16, 40, 40)
	CNN_2	4×4, 32	(32, 20, 20)
	CNN_3	3×3, 16	(16, 10, 10)
	FC_1	-	(784,)
action embedding	Embedding	-	(7056,)
action head	reshape to 2D	-	(84, 84)
	CNN_1	1×1, 8	(8, 84, 84)
	CNN_2	1×1, 8	(8, 84, 84)
	CNN_3	1×1, 1	(84, 84)
action merge	CNN_1	1×1, 1	(84, 84)

B. Related Work

Optimization-based Method for Placement. As a combinatorial optimization problem, researchers have tried different optimization-based methods to solve the chip placement task, which can be classified into partitioning-based, simulated annealing-based, and analytic-based methods.

Partitioning-based methods (Roy et al., 2006; Khatkhate et al., 2004) are based on the divide-and-conquer idea. They first divide the circuits into sub-circuits and assign each part to a sub-region on the chip. Then, these sub-circuits can be

Table 11: Hyper-parameters used in our experiments

Stage	Configuration	Value	Configuration	Value
circuit token extraction	dim of hidden layer training iterations num of node features	32 800 4	dim of latent variable learning rate	32 1e-2
pretraining	num of layers #L num of att heads #A batch size	6 8 32	hidden size #H num of tokens learning rate	128 1+2×256 6e-4
finetuning	temperature for sampling α entropy weight λ batch size	1e-2 0.5 32	replay buffer size expected entropy β learning rate	64 0.5 1e-4
standard cell placement	density weight in (phase 2) number of iterations (phase 2)	1e-4 300	density weight in (phase 3)	1e-3
computing hardware	CPU	AMD Ryzen 9 5950X	GPU	2× RTX 3090

continuously divided and assigned to divided sub-regions recursively. These methods are efficient but can hardly get high placement quality, especially for large-scale circuits.

Simulated annealing-based methods (Yang et al., 2000; Vashisht et al., 2020) start from a randomly generated placement solution and then try to move to a neighbor solution with a slight change. If the neighbor solution is better than the current solution, it uses the neighbor solution to replace it. If the neighbor solution is worse than the current solution, there is also a probability of decreasing with the iterative process to accept the neighbor solution. Such a solution can achieve improved placement quality but suffers from high computational inefficiency.

Analytical-based methods express the optimization problem as an analytical function of the coordinates of modules. Based on the analytical functions, these methods can be divided into quadratic methods (Viswanathan et al., 2007a;b; Kim et al., 2012; Kim & Markov, 2012; Brenner et al., 2015; Lin et al., 2013; Spindler et al., 2008) and non-quadratic methods (Chen et al., 2008; Lu et al., 2014; Cheng et al., 2018; Lin et al., 2020; Chan et al., 2006; Kahng et al., 2005; Gu et al., 2020). Quadratic methods use quadratic functions as objective functions and solve placement by Quadratic Programming (QP) (Nocedal & Wright, 2006). These methods are more applicable to problems where the search space is discrete, so they are still used in FPGA placement. On the contrary, in the non-quadratic methods, the objective functions are not quadratic and contain some non-linear functions, such as the logarithmic function. The advantage is that these functions can describe the placement target more precisely (Cheng et al., 2018; Lu et al., 2014). However, the corresponding optimization methods, such as stochastic gradient descent, will lead to a locally optimal solution.

Learning-based Method for Placement. With the development of deep learning techniques, researchers have proposed learning-based placement methods, which can be divided into reinforcement learning-based, supervised learning-based, and unsupervised learning-based methods.

All existing reinforcement learning-based methods belong to online methods. They formulate the placement as a sequential Markov decision process and decide the position of one module at each step. GraphPlace (Mirhoseini et al., 2021) uses the policy gradient framework (Sutton et al., 1999) to encode the current placement state by GCN-based model and generates the probability matrix of placement positions by de-convolution layers. After placing all hard macros, it uses the force-directed method (a quadratic method) (Spindler et al., 2008) to place the remaining soft macros and standard cells. After that, Jiang et al. (2021) verified that using the non-quadratic method DREAMPlace rather than the force-directed method could achieve better results. DeepPR (Cheng & Yan, 2021) and PRNet (Cheng et al., 2022) combine the CNN and GCN methods to encode the placement state and train the PPO model (Schulman et al., 2017) to learn the policy. However, they do not consider the actual sizes of macros, which means the reward function for HPWL is not accurate, and the overlaps cannot be avoided when placing macros. MaskPlace (Lai et al., 2022) introduces a series of visual representations of states in placement from the perspective of view, position, and wirelength. All of them are 2D images, where the view image is a high-level representation to describe the placement status from the human view. The position and wirelength images are low-level representations of overlapping positions and wirelength increases. However, it does not consider the topology information of circuits and previous states when making decisions by the CNN-based model, which tends to lead to a sub-optimal solution. Kim et al. also proposed an offline reinforcement learning method for the recap placement task, but it is for high-frequency

analog circuits, the scale of which is much smaller than the placement of large-scale digital circuits.

Supervised learning-based methods such as Flora (Liu et al., 2022a) and GraphPlanner (Liu et al., 2022b) considered that transforming the placement task into a sequential decision process makes itself complicated. On the contrary, they train the GAT (Veličković et al., 2017) and Variation GCN (Kipf & Welling, 2016a) models on a synthetic training dataset to predict the positions of modules as a kind of prediction task. However, the prediction results are hardly used directly because the non-overlapping constraint is not considered. Also, the synthetic data is still far from the real circuits.

Unsupervised learning-based methods are still in the early stages of research. Lu et al. (2022) proposed to use the GNN-based model to cluster modules and use the clustering results as soft constraints for the following placement solvers (*i.e.,* the solver will make the positions of modules under the same cluster as close as possible). Due to a lack of labeling information for clustering, the estimated objective function based on metrics such as congestion, timing, and power is not guaranteed to be accurate.

Algorithm 1 Generate circuit token HI(c_{test})

Input: Training data $\mathcal{M}_{\text{train}}^{[n]}$, including adjacency metrics $A^{[n]}$, node features $X^{[n]}$ of n chip circuits, inference model $q_\phi(X, A)$, and generative model $p(Z)$. Testing data $\mathcal{M}_{\text{test}}$ for the unseen circuit c_{test}.
for epoch $= 1$ **to** E **do**
  $Z^{[n]} = q_\phi(X^{[n]}, A^{[n]})$
  $\hat{A}^{[n]} = p(Z^{[n]})$
  loss $=$ BCEloss$(A^{[n]}, \hat{A}^{[n]})$
  $\phi = \phi - \partial \text{loss}/\partial\phi$
end for
$Z = q_\phi(X^{c_{\text{test}}}, A^{c_{\text{test}}})$ {Generate latent variables for N macros.}
HI$(c_{\text{test}}) = \sum_{i=1}^{N} z_i/N$ {Circuit token (chip representation) is the average pooling for all variables of macros}