Reinforcement Learning within Tree Search for Fast Macro Placement

Zijie Geng [1] **Jie Wang** [1 ✉] **Ziyan Liu** [1] **Siyuan Xu** [2] **Zhentao Tang** [2]
Mingxuan Yuan [2] **Jianye Hao** [2 3] **Yongdong Zhang** [1] **Feng Wu** [1]

Abstract

Macro placement is a crucial step in modern chip design, and reinforcement learning (RL) has recently emerged as a promising technique for improving the placement quality. However, existing RL-based techniques are hindered by their low sample efficiency, requiring numerous online rollouts or substantial offline expert data to achieve bootstrap, which are often impractical in industrial scenarios. To address this challenge, we propose a novel sample-efficient framework, namely **EfficientPlace**, for fast macro placement. EfficientPlace integrates a global tree search algorithm to strategically direct the optimization process, as well as a RL agent for local policy learning to advance the tree search. Experiments on commonly used benchmarks demonstrate that EfficientPlace achieves remarkable placement quality within a short timeframe, outperforming recent state-of-the-art approaches.

1. Introduction

Macro placement is a crucial step in modern chip design, as it significantly impacts the overall quality of the final chip (MacMillen et al., 2000; Markov et al., 2012). This task is essentially a large-scale optimization problem, which requires determining the positions of macros (i.e., rectangular circuit modules) on a chip canvas (i.e., a rectangular container) without any overlap (Wang et al., 2009). Due to the lengthy workflow of chip design, designers often rely on surrogate metrics that effectively reflect the final results to guide the optimization process in macro placement. Among these metrics, a commonly adopted metric is the half-perimeter wirelength (HPWL), which provides an approximation for the routing wirelength and is widely used to measure the placement quality (Rabaey et al., 2002; Lai et al., 2022; Shi et al., 2023). Despite the simplification of the surrogate metrics, the macro placement task remains challenging due to the vast design space, which can even significantly surpass that of sophisticated strategy games such as Go (Mirhoseini et al., 2021).

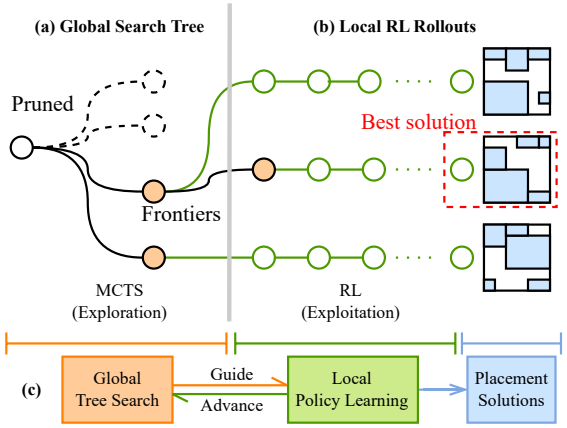


Figure 1. **The bi-level framework in EfficientPlace. (a) High-Level Global Search Tree.** It includes a search tree to explore the placement space. It identifies promising nodes (called frontiers), and prunes less valuable nodes, thus pinpointing potential areas for optimization. **(b) Low-Level Local RL Rollouts.** It employs a RL agent to exploit the identified promising nodes, promoting the evolution of solutions. **(c) Synergistic Bi-Level Interaction.** The global tree search strategically guides the local policy towards promising nodes, enhancing learning efficiency. Conversely, local policy learning advances the tree search by effectively exploiting these nodes, leading to the optimal placement solutions.

Existing macro placement methods mainly fall into two categories: optimization-based methods and reinforcement learning (RL)-based methods (Lai et al., 2023). Optimization-based methods employ traditional optimization algorithms, such as simulated annealing (SA) (Vashisht et al., 2020) and evolutionary algorithms (EA) (Shi et al., 2023) to directly address the large-scale optimization problem, exploring the design space to identify near-optimal solutions. However, they often necessitate a post-processing

This work was done when Zijie Geng was an intern at Huawei. [1]CAS Key Laboratory of Technology in GIPAS & MoE Key Laboratory of Brain-inspired Intelligent Perception and Cognition, University of Science and Technology of China [2]Noah's Ark Lab, Huawei, China [3]Tianjin University, Tianjin, China. Correspondence to: Jie Wang <jiewangx@ustc.edu.cn>.

Proceedings of the 41st International Conference on Machine Learning, Vienna, Austria. PMLR 235, 2024.

step to accommodate the non-overlapping constraint, and they generally lack the capacity to learn from past experiences, thus remaining unsatisfactory in quality or efficiency. In recent research, macro placement has been formulated as a Markov Decision Process (MDP), where the macro positions are determined sequentially (Mirhoseini et al., 2021). Reinforcement learning (RL) has emerged as a promising technique for this task due to its ability to continuously improve performance based on feedback from the environment through trial and error (Cheng & Yan, 2021; Cheng et al., 2022; Lai et al., 2022). However, RL methods typically require a significant number of rollouts to achieve bootstrap, resulting in low training efficiency (Lai et al., 2023).

This efficiency issue poses a significant hinder, especially in industrial chip design when engineers require fast placement strategies to streamline the workflow across multiple design iterations. Early design stages like logical synthesis and floorplanning rely on feedback from later stages, calling for a placement strategy with both high quality and efficiency. This presents challenges for RL-based methods (Shi et al., 2023). The recent development, ChiPFormer (Lai et al., 2023), has attempted to address the challenge of efficiency through an offline RL strategy. However, its reliance on a substantial amount of expert data, which is often scarce in practical settings, limits its applications. Moreover, the substantial variation among different chip domains necessitates extensive fine-tuning steps of ChiPFormer, hindering its practicality in industrial scenarios.

To address the aforementioned challenge in RL-based chip placement, we propose a novel sample-efficient framework, namely **EfficientPlace**, for fast macro placement. Its efficiency is derived from two aspects. **Bi-level Learning-Inside-Optimization Framework.** Our basic observation is the duality inherent in the macro placement task. On one hand, it is an optimization problem, aimed to search for the optimal solutions within a vast design space. On the other hand, the vastness of the search space calls for deep reinforcement learning's ability to learn and generalize through trial and error. EfficientPlace leverages this duality with a "learning-inside-optimization" framework. It facilitates the accumulation of online experiences within the search episode, creating a bi-level architecture that merges learning with optimization. An shown in Figure 1, it encompasses a global tree search and a local policy learning. At the higher level, Monte Carlo Tree Search (MCTS) is used to strategically navigate the search process. It utilizes a novel mechanism of rolling frontiers to guide the agent to focus on and exploit valuable states, thus enhancing the optimization of elite solutions. The lower level leverages the RL agent's learning and generalization capabilities, promoting efficient exploration across the expansive search space. This bi-level framework generates a dynamic synergy: direct optimization guided by MCTS and adaptive learning driven by RL.

Sample-Efficient RL Agent Design. To further improve sample efficiency, we make two critical enhancements on the RL agent design. First, the agent leverages wiremask proposed by Lai et al. (2022)—a matrix that quantifies the HPWL increment at each canvas position—to narrow down the exploration space and guide the decision process. Second, the RL agent incorporates a U-net architecture, which allows for a high-resolution control, even on a canvas up to 512×512 grid size, surpassing previous RL-based methods in precision. This supports a fine-grained placement, further improving the placement quality.

Extensive experiments on commonly used benchmark circuits demonstrate that EfficientPlace outperforms recent state-of-the-art (SOTA) methods, including the offline RL-based approach ChiPFormer (Lai et al., 2023) and the optimization-based approach WireMask-EA (Shi et al., 2023). Notably, EfficientPlace achieves the best Half-Perimeter Wire Length (HPWL) within just $1,000$ steps from scratch, costing 2.2 hours on each chip circuit on average. It outperforms ChiPFormer, even with a pre-training and additional $2,000$ steps of fine-tuning, and WireMask-EA, which takes 6.9 hours on average. We further conduct extensive ablation studies to validate the significance of each components in our method. We highlight this work's contributions to the research community as follows.

- **The Novel Framework.** We propose a new "learning-inside-optimization" framework to leverage the duality of macro placement. By integrating the strengths of both paradigms, i.e., direct search and adaptive learning, our approach significantly boosts the efficiency of macro placement.
- **Superior Performance.** Our proposed model, EfficientPlace, sets new baselines in HPWL and training efficiency, outperforming recent state-of-the-art methods on the widely recognized benchmarks.
- **Comprehensive Analysis.** We conduct extensive experiments to investigate the impact of different optimization and learning strategies and the components on macro placement. This analysis provides essential insights that pave the way for future technological advancements in this field.

2. Related Work

Optimization-based Methods Macro placement, as a complex combinatorial optimization problem, has been tackled with various optimization-based methods, including analytical methods, partition-based methods, and black-box-optimization (BBO) methods. Analytical methods formulate the optimization objective as an analytical function of module coordinates. Then it can be efficiently solved using techniques like quadratic programming (Kahng et al.,

2005; Viswanathan et al., 2007a;b; Spindler et al., 2008; Chen et al., 2008; Kim et al., 2012; Kim & Markov, 2012; Cheng et al., 2018) and direct gradient descent (Lin et al., 2019; 2020; Gu et al., 2020; Liao et al., 2022). Partitioning-based methods, following a divide-and-conquer strategy, split circuits into sub-circuits for assignment to chip sub-regions (Roy et al., 2006; Khatkhate et al., 2004). These methods, though efficient, often relax the non-overlap constraint during optimization, thus struggling to consistently derive high-quality placements. Another perspective views macro placement as a BBO problem. Simulated annealing (SA) is a commonly used algorithm for such BBO problems, which creates new solutions through genotype perturbation and phenotype evaluation (Kirkpatrick et al., 1983; Sherwani, 2012; Ho et al., 2004; Shunmugathammal et al., 2020; Vashisht et al., 2020). Solution representations like sequence pair (Murata et al., 1996) and B*-tree are proposed to map genotypes to placement solutions (Chang et al., 2000). Though achieving improved quality, these methods are computationally inefficient. A recent advancement is a WireMask-BBO (Shi et al., 2023), which employs a wiremask-guided greedy strategy for post-processing, thus enhancing the efficiency of BBO algorithms in the genotype space. It supports different BBO algorithms and finds that WireMask-EA, which applies evolutionary algorithms for coordinate swapping, achieves the best overall performance and outperforms previous RL-based methods.

Learning-based Methods As the scale of modern Very-Large-Scale Integration (VLSI) systems expands, classical optimization-based approaches are facing increasing challenges. Researchers have been exploring learning-based methods, particularly RL-based methods, for better placement quality. GraphPlace (Mirhoseini et al., 2021) first models Macro Placement as a Reinforcement Learning (RL) problem. Subsequently, DeepPR (Cheng & Yan, 2021) and PRNet (Cheng et al., 2022) establish a streamlined pipeline encompassing macro placement, cell placement, and routing. These methods treat density as a soft constraint, and so they may violate non-overlap constraint during training. MaskPlace (Lai et al., 2022) approaches placement problem at a pixel level, employing a position mask to avoid overlap, as well as a wiremask—which represents the increment of HPWL when placing next macro at each grid of the canvas—as a visual input, and using dense reward to boost the efficiency. ChiPFormer (Lai et al., 2023) is the first offline RL method. It is pre-trained on various chips via offline RL and then fine-tuned on unseen chips for better efficiency. Despite the achievements, RL-based methods continue to face the issue of sub-optimal sample efficiency.

Reinforcement Learning and MCTS Reinforcement learning (RL) has been widely adopted in many decision-making scenarios (Yang et al., 2022; Wang et al., 2022; 2023b; Zhang et al., 2023; Chen et al., 2024), especially in the fields of combinatorial optimization (Wang et al., 2023a; 2024a; Ling et al., 2024; Li et al., 2024; Geng et al., 2024) and chip design (Wang et al., 2024b;c). The integration of RL and Monte Carlo Tree Search (MCTS) has led to a series of breakthroughs (Browne et al., 2012), including AlphaGo (Silver et al., 2016; 2017), MuZero (Schrittwieser et al., 2020; Ye et al., 2021), AlphaTensor (Fawzi et al., 2022), and AlphaDev (Mankowitz et al., 2023). These methods primarily employ MCTS to identify optimal decisions and guide the RL agent for policy improvement, which necessitates numerous rollouts. We propose a new framework that changes the role of the tree search. In our framework, tree search serves as a global guide rather than a local policy optimizer. This approach prioritizes finding an optimal solution over learning a comprehensive policy, offering a specialized solution to macro placement's demands.

3. Preliminaries

Macro Placement The ultimate goal of chip design is to optimize the power, performance, and area (PPA) metrics of the final chip. This design workflow is divided into several stages, such as placement and routing, with the placement stage subdivided into macro and cell placement. A variety of heuristic metrics have been used to guide these stages to contribute positively to the chip's final quality, such as HPWL and congestion. The Half Perimeter Wire Length (HPWL) serves as an efficient estimator of wirelength, which is a crucial metric for indicating the chip performance but can be assessed only post-routing. Congestion indicates the routability of placement outcomes and affects the manufacturing process directly. Among these metrics, HPWL stands out as a widely accepted metric due to its computational efficiency and has been extensively utilized in prior studies (Lai et al., 2022; 2023; Shi et al., 2023) to assess placement quality. Additionally, Shi et al. (2023) has demonstrated that optimizing HPWL could also improve other metrics, such as congestion. Therefore, in alignment with the prior works, we adopt HPWL as the primary optimization objective.

HPWL HPWL is defined as the sum of the half-perimeters of the bounding boxes for all nets in the chip circuit. Formally, it is expressed as:

$$\text{HPWL} = \sum_{e_i \in E} (w_i + h_i), \tag{1}$$

where E denotes the set of all nets in the chip, $w_i = \max_{v_j \in e_i} x_j - \min_{v_j \in e_i} x_j$ and $h_i = \max_{v_j \in e_i} y_j - \min_{v_j \in e_i} y_j$ denote the width and height of net e_i, respectively, with v_j representing pins in the nets. The calculation HPWL is visually detailed in Figure 7 in Appenix A.3.

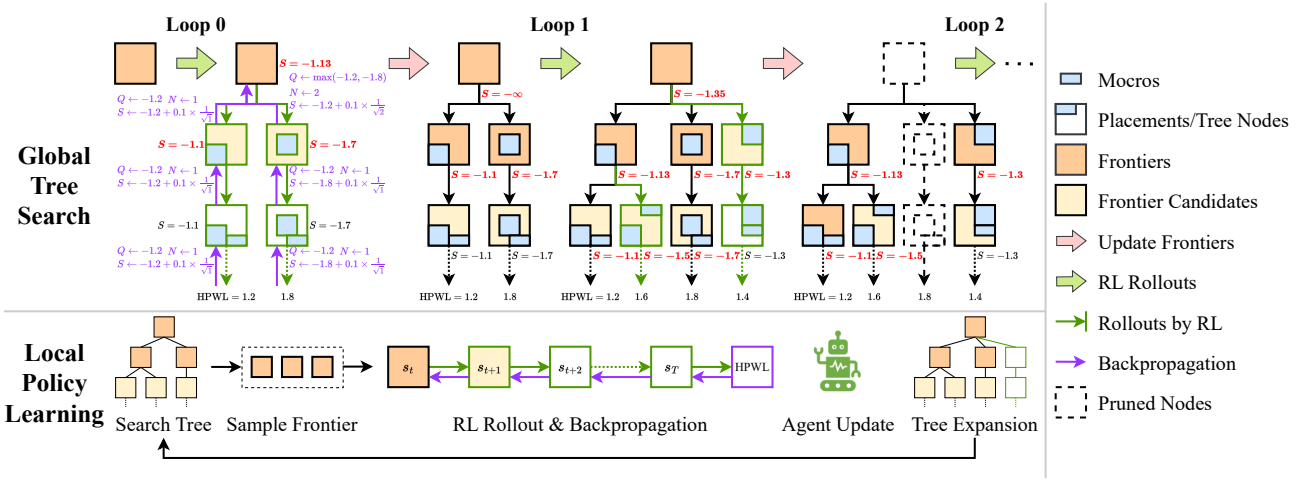


Figure 2. **Method Overview. (1) Global Tree Search.** We construct a search tree where each node represents a placement state. We dynamically manage a set of "frontiers", i.e., nodes that represent current states of focus. In each loop, multiple rollouts are executed with a frontier as the initial state, serving to expand the tree and inform its evolution through backpropagation. The frontiers are updated in each loop, and nodes not deserved revisiting are thus pruned. **(2) Local Policy Learning.** We employ a reinforcement learning (RL) agent to execute the rollouts. It focuses on exploiting the frontier nodes and drive the tree's expansion. The RL agent is trained during the search process, enhancing its ability to conduct effective local searches within the broader tree structure.

Notations Our objective is to determine optimal positions for a set of T macros, $\{M_0, M_1, \cdots, M_{T-1}\}$, on an $N \times N$ chip canvas. Previous works have proposed different heuristics to determine the placement orders of the macros (Mirhoseini et al., 2021; Lai et al., 2022). In this work, we simply order the macros in descending order by their areas, as empirically larger-area macros significantly influence the overall layout results (Shi et al., 2023). Let $\mathcal{A}$ be the set of position cells, with $|\mathcal{A}| = N^2$. Each macro M_i will be assigned a position $a_i = (x_i, y_i) \in \mathcal{A}$, where (x_i, y_i) represents the xy-coordinate position of the bottom-left corner of M_i. The position is denoted as a_i because we can treat the grid cell position as a placement action. The state of the canvas at any step t is uniquely defined by the macro positions that have been determined up to the current step, denoted as $s_t = (a_0, a_1, \cdots, a_{t-1})$. We denote the empty canvas as s_0, and denote the valid state space at step t is denoted as $\mathcal{S}_t$. With the above notations, the macro placement task can be described as:

$$\min_{s \in \mathcal{S}_T} \text{HPWL}(s). \tag{2}$$

4. Our Approach

This section is organized as follows. We first elaborate the motivation behind integrating learning within optimization for macro placement (Section 4.1). We then present the structure of the proposed bi-level framework, showcasing the synergy between global tree search and local policy learning (Section 4.2). Finally, we delve into the architecture of our efficient RL agent (Section 4.3).

4.1. Motivation

We begin by revisiting the paradigms of optimization-based and learning-based methods in the context of macro placement. Optimization methods, such as SA that introduces perturbations to the existing solution and WireMask-EA that generates new solutions through macro position swaps, primarily focus on identifying a singular, optimal solution s_T^* within the solution space $\mathcal{S}_T$. Their main strength lies in their direct approach and consistent refinement of the best solution. However, the lack of learning capability for handling unseen states limits their sample efficiency in the vast search spaces. In contrast, learning-based approaches mainly aim to develop decision-making policies for positioning macros at various intermediate states. They leverage neural networks' generalization capabilities to achieve a balance between exploration and exploitation. This feature is particularly beneficial in navigating unexplored states. However, existing RL methods start independent rollouts from an empty canvas, which may lead to biases towards sub-optimal states and redundant computation.

Macro placement is characterized by a dual nature. On one hand, it involves finding the best solution for specific circuits, a task well-suited for optimization methods. On the other hand, the vastness and complexity of the design space call for the adaptive learning capabilities of RL. The strength of RL in assimilating past insights, navigating through complex spaces and making informed decisions in new situations is valuable in such scenarios. This offers a motivation to integrate the learning aspect of RL into the oriented optimization process.

Algorithm 1 EfficientPlace

Input: Circuit netlist, search tree $\mathcal{T}$, RL agent $\pi_{\boldsymbol{\theta}}(a_t|s_t)$
Parameters: Number of loops K_{TS}, K_{RL}
Output: Optimized placement solution
Initialize $\mathcal{T}$.add_node(s_0), $\mathcal{F} \leftarrow \{s_0\}$
for $i = 1$ **to** K_{TS} **do**
  for $j = 1$ **to** K_{RL} **do**
    Select a frontier node $s \sim \mathcal{F}$
    Execute a rollout from state s with $\pi_{\boldsymbol{\theta}}$
    Expand $\mathcal{T}$ with new states in the rollout
    Backpropagate the rollout results
    Update the RL agent $\pi_{\boldsymbol{\theta}}$
  end for
  Update the frontiers $\mathcal{F}$
end for
Return: The best placement solution s_T^* in $\mathcal{T}$

Algorithm 2 Updating Frontiers

Input: Search tree $\mathcal{T}$, the current frontiers $\mathcal{F}$
Parameters: Frontier set capacity $K_{\mathcal{F}}$, coefficient α
Output: Updated frontiers $\mathcal{F}$
$\mathcal{F}_{\text{cand}} \leftarrow \bigcup_{s\in\mathcal{F}} \mathcal{C}(s) \setminus \mathcal{F}$.
$s_1 \leftarrow \arg\min_{s\in\mathcal{F}} Q(s)$, $s_2 \leftarrow \arg\max_{s\in\mathcal{F}_{\text{cand}}} Q(s)$
while $S(s_1) < S(s_2)$ or $|\mathcal{F}| < K_{\mathcal{F}}$ **do**
  $s \leftarrow s_2$.parent, $Q(s) \leftarrow \max_{s'\in\mathcal{C}(s)\setminus\{s_2\}} Q(s')$
  $S(s) \leftarrow Q(s) + \alpha \cdot U(s)$
  while $s \neq$ root **do**
    $s \leftarrow s$.parent
    $Q(s) \leftarrow \max_{s'\in\mathcal{C}(s)} Q(s'), S(s) \leftarrow Q(s)+\alpha\cdot U(s)$
  end while
  $\mathcal{F} \leftarrow \mathcal{F} \cup \{s_2\}\ if\ |\mathcal{F}| < K_{\mathcal{F}}\ else\ \mathcal{F} \setminus \{s_1\} \cup \{s_2\}$
  $\mathcal{F}_{\text{cand}} \leftarrow \mathcal{F}_{\text{cand}} \setminus \{s_2\}$
  $s_1 \leftarrow \arg\min_{s\in\mathcal{F}} Q(s)$, $s_2 \leftarrow \arg\max_{s\in\mathcal{F}_{\text{cand}}} Q(s)$
end while
Return: frontiers $\mathcal{F}$

4.2. EfficientPlace: The Bi-Level Framework

Building on the above discussion, we introduce **EfficientPlace**, a bi-level framework that merges the precision of tree search algorithms with RL's flexibility. The model structure is outlined in Figure 2, with the algorithm detailed in Algorithm 1. At the higher level, a global tree search is employed for strategic guidance, focusing on identifying and leveraging promising states. The lower level is to harness the learning and generalization of RL for efficient exploration. This synergy between the two levels creates a dynamical robust approach to macro placement, encapsulating the direct search guided by MCTS and adaptive learning driven by RL.

Global Search Tree We construct the search tree $\mathcal{T}$ as follows. Each node in $\mathcal{T}$ corresponds to a potential state $s_t \in \mathcal{S}_t$. The placement process initiates from the root node s_0, representing an empty canvas. As the tree develops, each newly encountered, unexplored state s_t is added as a new node, progressively expanding $\mathcal{T}$. We denote the children of a node s as $\mathcal{C}(s)$, and the set of its descendant leaf nodes as $\mathcal{P}(s)$. The expansion progresses towards the leaf nodes $s_T \in \mathcal{S}_T$, each signifying a specific placement outcome.

Frontiers & Node Selection We employ a beam-search strategy for node selection, focusing on a set of nodes called "frontiers", denoted as $\mathcal{F}$. Initially, $\mathcal{F}$ is set to $\{s_0\}$ and is dynamically updated in a rolling forward manner. The capacity of $\mathcal{F}$ is $K_{\mathcal{F}}$, ensuring that $|\mathcal{F}| \leq K_{\mathcal{F}}$ throughout the search. This constraint prompts us to concentrate on the most promising states based on previous evaluations. In each rollout, a frontier node s is randomly selected from $\mathcal{F}$, promoting diverse engagement by the RL agent and reducing policy learning biases.

Local Policy Learning & Tree Expansion Each RL rollout begins from a selected frontier node s, using it as the initial state. After each individual rollout, the outcomes are "backed up" (i.e., backpropagated) through the traversed nodes to update their statistics. We record the visit count $N(s)$ and the minimum HPWL values for each visited node. We define $Q(s)$ as the best evaluation of a node s, i.e.,

$$Q(s) = \max_{s_T\in\mathcal{P}(s)} -\text{HPWL}(s_T). \tag{3}$$

We use the proximal policy optimization (PPO) algorithm (Schulman et al., 2017) to update the RL agent.

Updating Frontiers After a number of rollouts, we update the frontiers before proceeding to the next loop. This update mechanism is inspired by optimization methods that maintain a pool of the best solutions. The goal is to keep nodes in $\mathcal{F}$ that are likely to yield fruitful exploration. Given the current $\mathcal{F}$, we define the set of frontier candidates as all children of the nodes in $\mathcal{F}$, i.e.,

$$\mathcal{F}_{\text{cand}} = \bigcup_{s\in\mathcal{F}} \mathcal{C}(s) \setminus \mathcal{F}. \tag{4}$$

Inspired by the UCT algorithm (Browne et al., 2012), we define the scores of a state s as:

$$S(s) = Q(s) + \alpha \cdot U(s), \tag{5}$$

where $U(s) = \frac{1}{\sqrt{N(s)}}$ is the bonus to encourage maintaining the current frontiers if they are under-exploited, and α is the coefficient. Nodes with both higher Q values and higher scores in $\mathcal{F}_{\text{cand}}$ are then selected as new frontiers. Notice that the frontiers are not always those nodes with

maximum depth. The specifics of this updating process are elaborated in Algorithm 2. The following theorem shows the improvement attribute of our tree search algorithm.

Theorem 1. *Suppose we are at the k^{th} iteration, with the set of frontiers $\mathcal{F}_k$ where $|\mathcal{F}_k| = K_{\mathcal{F}}$, the RL policy π_k, and the evaluation Q_k. After conducting multiple RL rollouts, we obtain an improved policy π_{k+1}. We assume that every $s \in \mathcal{F}_k$ is visited, and that $v^{\pi_{k+1}}(s) \geq v^{\pi_k}(s)$ for every $s \in \mathcal{F}_k$, where v^{π} represents the value function. This iteration results in the updated Q_{k+1} and $\mathcal{F}_{k+1}$. Then we have*

$$\mathbb{E}_{s\sim\mathcal{F}_{k+1}}[Q_{k+1}(s)] \geq \mathbb{E}_{s\sim\mathcal{F}_k}[Q_k(s)], \tag{6}$$

and

$$\mathbb{E}_{\pi_{k+1}}[\mathbb{E}_{s\sim\mathcal{F}_{k+1}}[Q_{k+1}(s)]] \geq \mathbb{E}_{s\sim\mathcal{F}_k}[v^{\pi_{k+1}}(s)]. \tag{7}$$

The proof can be found in Appendix A. We can conclude from this theorem that our search algorithm consistently enhances the quality of the solutions over iterations, and that it outperforms the pure RL method in the sense of expectation.

4.3. RL Agent Architecture

In alignment with the previous work MaskPlace (Lai et al., 2022) and ChiPFormer (Lai et al., 2023), our RL agent takes visual inputs. These inputs include an image of the current canvas, the position mask that identifies valid action positions, and the wiremask, which indicates the HPWL increase for placing a macro at each canvas position. The methodologies for computing position masks and wiremasks are illustrated in Figure 3. We identify two critical components that effectively boost the efficiency of macro placement, which are different from previous works.

Wiremask for Reducing Search Space The concept of wiremask was introduced by Lai et al. (2022) as the visual inputs to the neural networks. Shi et al. (2023) then employed wiremask to devise a greedy policy to guide the BBO algorithms. Similar to them, we restrict actions to the grid areas with the minimal HPWL increment, which narrows down the search space, thereby significantly enhancing the training efficiency and the placement quality.

U-Net for Fine-Grained Control Precise control over the canvas grid is essential for accurate macro placement. To achieve a fine-grained control without sacrificing efficiency, we utilize a U-Net architecture as the policy network. The U-Net's left side serves as the encoding stage, processing visual inputs and integrating them with netlist information and timestep embeddings. The right side, functioning as the decoder, outputs policy probabilities. The left-side features are fused into the right-side layers so that the shallow-layer input information is effectively conveyed to the decoder,

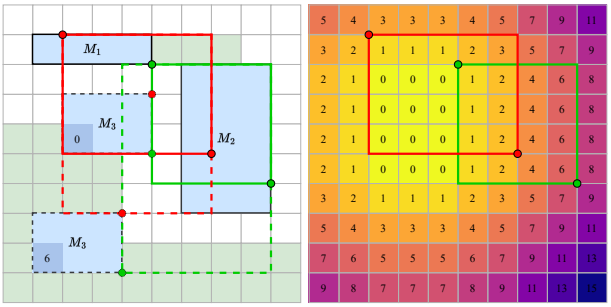


Figure 3. **Illustation of the position mask and wiremask calculation.** In this figure, M_1 and M_2 represent macros that have already been placed, and M_3 represents the next macro to place. The position mask is a matrix to identify feasible grid positions for placement, which are marked in in green. The red and green solid boxes represent the bounding boxes of two nets. The wiremask is a matrix to quantify the increment of HPWL that would result from placing M_3 in each specific grid position.

facilitating more precise control over placement decisions. Moreover, the decoder employs bilinear upsample blocks to allow the network to produce close values for neighboring grid points. This design empowers the RL agent to leverage the relative positioning of grid lattices, thus enhancing the precision and efficiency of exploration. With the above model structure, we achieve efficient training even on large canvases up to 512×512, leading to improvements in HPWL performance. The detailed network structure is in Figure 8.

5. Experiments

Benchmarks and Settings. We evaluate EfficientPlace and compare it with several recent state-of-the-art macro placement methods, including GraphPlace (Mirhoseini et al., 2021), DeepPR (Cheng & Yan, 2021), MaskPlace (Lai et al., 2022), ChiPFormer (Lai et al., 2023), SA (Cheng et al., 2023), and WireMask-EA (Shi et al., 2023). Among these methods, GraphPlace, DeepPR, MaskPlace and ChiPFormer are RL-based methods, and SA and WireMask-EA are BBO-based methods. All the experiments are conducted on a single machine with NVidia GeForce GTX 3090 GPUs and Intel(R) Xeon(R) E5-2667 v4 CPUs 3.20GHz. Following the previous studies (Lai et al., 2022; 2023; Shi et al., 2023), we evaluate these methods on the commonly used ISPD2005 benchmark (Nam et al., 2005), which comprises eight datasets: *adaptec1-4* and *bigblue1-4*. Further details on these circuits and the experimental setup are available in Appendix A.3. Our code is available at `https://github.com/MIRALab-USTC/AI4EDA-EfficientPlace.git`.

Main Results. Table 1 presents results of macro placements using different approaches. We evaluate each ap-

Table 1. HPWL values ($\times 10^5$) obtained from seven compared methods on eight chip circuits. The results of the SA algorithms are derived by running the code from Cheng et al. (2023). The results of other baseline methods are from Shi et al. (2023) and Lai et al. (2023). As WireMask-EA does not provide the results on *bigblue2*, we run their released code with the grid size as 128. Results are from five independent runs with different random seeds, and we report the means and standard deviations (mean±std). Numbers after method names denote the steps required to achieve the reported results. We also report the runtime of EfficientPlace and WireMask-EA for comparison. We mark **the best** results in bold and underline the second-best results.

Method	adaptec1	adaptec2	adaptec3	adaptec4	bigblue1	bigblue2	bigblue3	bigblue4
GraphPlace (50k)	30.01±2.98	351.71±38.20	358.18±13.95	151.42±9.72	10.58±1.29	14.78±0.95	357.48±47.83	440.70±15.95
DeepPR (3k)	19.91±2.13	203.51±6.27	347.16±4.32	311.86±56.74	23.33±3.65	11.38±0.20	430.48±12.18	433.90±5.26
MaskPlace (3k)	7.62±0.67	75.16±4.97	100.24±13.54	87.99±3.25	3.04±0.06	5.75±0.11	90.04±4.83	103.26±2.69
ChiPFormer (2k)	6.62 ±0.05	67.10±5.46	76.70±1.15	68.80±1.59	2.95±0.04	5.44±0.10	72.92±2.56	102.84±0.15
SA (1M)	19.24±1.42 (10.18h)	102.84 ± 5.83 (10.37h)	140.34 ± 7.48 (15.87h)	148.57 ± 23.08 (19.87h)	5.92 ± 0.98 (10.25h)	12.09 ± 0.42 (5.46h)	270.46 ± 43.76 (19.65h)	248.38 ± 12.25 (19.55h)
WireMask-EA (1k)	6.15±0.15 (4.78h)	64.38±4.43 (4.18h)	58.18±1.04 (4.60h)	59.52±1.71 (7.96h)	2.15±0.01 (2.30h)	5.14±0.06 (0.37h)	59.85±3.39 (17.48h)	77.54±0.67 (13.92h)
EfficientPlace (0.5k)	6.04±0.08 (0.43h)	47.04±1.44 (0.51h)	57.18±1.17 (0.95h)	59.47±0.58 (2.06h)	2.15±0.01 (0.71h)	4.79±0.17 (0.28h)	62.49±2.95 (1.90h)	78.72±1.10 (1.76h)
EfficientPlace (1k)	**5.94±0.04** (0.84h)	**46.79±1.60** (0.83h)	**56.35±0.99** (1.87h)	**58.47±1.61** (3.81h)	**2.14±0.01** (1.40h)	**4.67±0.11** (0.53h)	**58.38±0.54** (3.71h)	**76.63±1.02** (3.62h)

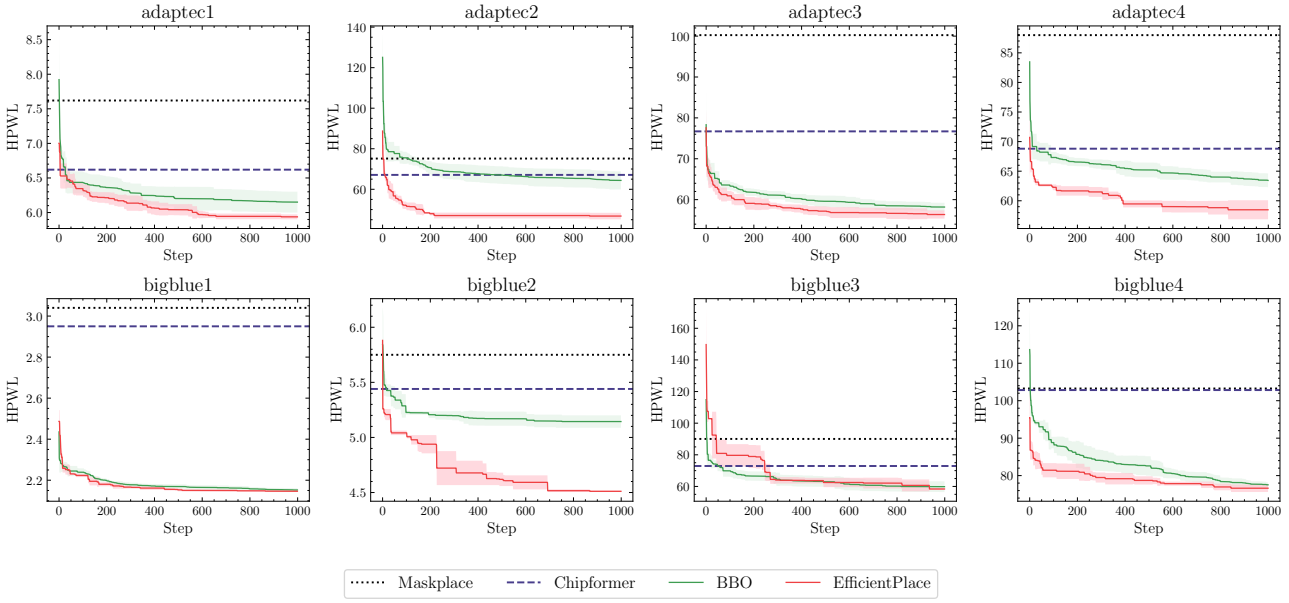


Figure 4. HPWL ($\times 10^5$) vs. run steps of EfficientPlace and WireMask-EA. The shaded region represents the standard error derived from 5 independent runs. For WireMask-EA, we run their released code to produce the results.

proach over five independent runs with different random seeds, and report the mean and variance of HPWL achieved. EfficientPlace consistently achieves the lowest HPWL values across all chips within $1,000$ steps, averaging 2.2 hours on per chip. This performance surpasses that of MaskPlace trained with $3,000$ steps, and ChiPFormer with its pre-training and additional $2,000$ steps of fine-tuning. It also outperforms WireMask-EA, which takes 6.9 hours per chip. Moreover, EfficientPlace achieves second-best results within merely 500 steps on most of the chips, further demonstrating its remarkable sample efficiency.

Runtime & Step Analysis. As WireMask-EA is the very recent state-of-the-art approach, we further compare EfficientPlace with WireMask-EA by analyzing their HPWL progression over run steps in Figure 4. Following Shi et al. (2023), we use the best HPWL value reached by each step for comparison. We also present the HPWL trend over the wall clock time in Figure 9 in Appendix B. The results demonstrate that Efficient outperforms WireMask-EA on both sample efficiency and time efficiency.

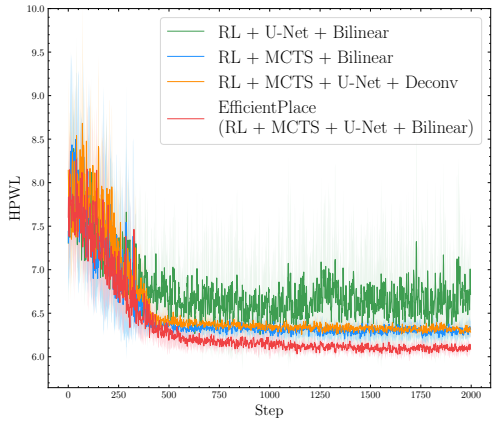


Figure 5. **Ablation Study on Model Components.** We test EfficientPlace with different configurations on the *adaptec1* dataset, and present HPWL vs. training step.

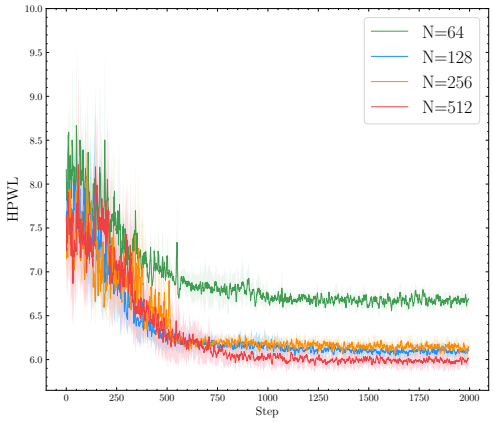


Figure 6. **Comparison Study on Grid Sizes.** We test EfficientPlace with different grid resolutions on the *adaptec1* dataset, and present HPWL vs. training step.

Congestion Results. We further investigate the congestion results, using the RUDY algorithm (Spindler & Johannes, 2007) to assess the congestion values. The detailed settings and results can be found in Appendix B.3. Though the primary object of our placer is to minimize the HPWL, these results demonstrate that such optimization not only optimizes HPWL values but also positively affects the congestion result, aligning the findings from Shi et al. (2023).

Mixed-size Placement. We then extend our analysis to mixed-size placement (incorporating both macros and standard cells) to illustrate how improved macro placement impacts subsequent design stages. Specifically, we use EfficientPlace for macro placement and then employ DREAMPlace (Lin et al., 2019) for cell placement. We report the full placement HPWL results and provide the results visualization in Appendix B.2. The detailed settings and results can be found in Appendix B.3. These results also demonstrate that such optimization not only improves macro placement but also has a positive impact on subsequent cell placement.

Ablation Study on Model Components. We conduct extensive ablation studies to investigate the impact of each component in our approach, including the global tree search algorithm, the U-Net policy architecture, and the bilinear upsampling technique. Results are in Figure 5. We test the performance on the `ISPD2005 adaptec1` dataset with the following different configurations. **(1) RL + U-Net + Bilinear:** Here, we maintain the RL agent identical to that in EfficientPlace but exclude the global tree search. Results show that the absence of tree search leads to notable performance oscillation, underscoring the tree search's significance in stabilizing training and promoting convergence. **(2) RL + MCTS + Bilinear:** We drop the connection between the U-Net's encoder and decoder, resulting in diminished results, which demonstrates the importance of U-Net structure for a more precise control. **(3) RL + MCTS + U-Net + Deconv:** Replacing bilinear upsampling with deconvolutional layers, despite more parameters, results in inferior performance. This is because the bilinear upsampling produces closer action probabilities for adjacent grids, effectively capturing the spatial relationship between actions, thus achieving a more efficient exploration. **(4) RL + MCTS + U-Net + Bilinear:** This combination represents EfficientPlace's full configuration and achieves the best performance, validating the effectiveness of our approach.

Comparison Study on Grid Sizes. EfficientPlace's design allows for precise control through finer grid resolutions. To showcase the benefits of increased grid granularity, we performed comparative analyses across different grid sizes N. Figure 6 illustrates EfficientPlace's performance on `ISPD2005` *adaptec1* across varying grid sizes. The results demonstrate that finer grids facilitate better learning curves, boost sample efficiency, and lead to lower HPWL metrics, showing the advantages of a more detailed grid approach.

6. Limitations, Outlook, and Conclusion

6.1. Limitation

We propose EfficientPlace for fast macro placement, while it mainly focuses on optimizing HPWL without explicitly considering more surrogate metrics. Future works will aim to incorporate additional metrics like dataflow and regularity for better alignment with the final PPA metrics.

6.2. Outlook

Our introduction of the "learning-inside-optimization" framework, recognizing the dual aspects of macro placement, sets the stage for further exploration in related domains. We expect this novel approach would inspire further research and applications in similar challenges.

6.3. Conclusion

In this papre, we propose EfficientPlace for fast macro placement. It harnesses the strengths of optimization and learning based methods to significantly boost sample efficiency. Experiments demonstrate that EfficientPlace outperforms recent state-of-the-art macro placement methods.

Acknowledgements

This work was supported in part by National Key R&D Program of China under contract 2022ZD0119801, National Nature Science Foundations of China grants U23A20388, 62021001, U19B2026, and U19B2044. This work was supported in part by Huawei as well. We would like to thank all the anonymous reviewers for their insightful comments.

Impact Statement

This paper introduces a "learning-inside-optimization" framework, and proposes EfficientPlace for fast macro placement. Experiments demonstrate that our approach exhibits great potential in benefiting the chip design area, accelerating the development of modern EDA tools.

References

Browne, C. B., Powley, E., Whitehouse, D., Lucas, S. M., Cowling, P. I., Rohlfshagen, P., Tavener, S., Perez, D., Samothrakis, S., and Colton, S. A survey of monte carlo tree search methods. *IEEE Transactions on Computational Intelligence and AI in games*, 4(1):1–43, 2012.

Chang, Y.-C., Chang, Y.-W., Wu, G.-M., and Wu, S.-W. B*-trees: A new representation for non-slicing floorplans. In *Proceedings of the 37th Annual Design Automation Conference*, pp. 458–463, 2000.

Chen, R., Liu, X.-H., Liu, T.-S., Jiang, S., Xu, F., and Yu, Y. Foresight distribution adjustment for off-policy reinforcement learning. In *Proceedings of the 23rd International Conference on Autonomous Agents and Multiagent Systems*, pp. 317–325, 2024.

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*, 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*, 38(9):1717–1730, 2018.

Cheng, C.-K., Kahng, A. B., Kundu, S., Wang, Y., and Wang, Z. Assessment of reinforcement learning for macro placement. In *Proceedings of the 2023 International Symposium on Physical Design*, pp. 158–166, 2023.

Cheng, R. and Yan, J. On joint learning for solving placement and routing in chip design. *Advances in Neural Information Processing Systems*, 34:16508–16519, 2021.

Cheng, R., Lyu, X., Li, Y., Ye, J., Hao, J., and Yan, J. The policy-gradient placement and generative routing neural networks for chip design. *Advances in Neural Information Processing Systems*, 35:26350–26362, 2022.

Fawzi, A., Balog, M., Huang, A., Hubert, T., Romera-Paredes, B., Barekatain, M., Novikov, A., R Ruiz, F. J., Schrittwieser, J., Swirszcz, G., et al. Discovering faster matrix multiplication algorithms with reinforcement learning. *Nature*, 610(7930):47–53, 2022.

Geng, Z., Li, X., Wang, J., Li, X., Zhang, Y., and Wu, F. A deep instance generative framework for milp solvers under limited data availability. *Advances in Neural Information Processing Systems*, 36, 2024.

Gu, J., Jiang, Z., Lin, Y., and Pan, D. Z. Dreamplace 3.0: Multi-electrostatics based robust vlsi placement with region constraints. In *Proceedings of the 39th International Conference on Computer-Aided Design*, pp. 1–9, 2020.

Ho, S.-Y., Ho, S.-J., Lin, Y.-K., and Chu, W.-C. An orthogonal simulated annealing algorithm for large floorplanning problems. *IEEE transactions on very large scale integration (VLSI) systems*, 12(8):874–877, 2004.

Kahng, A. B., Reda, S., and Wang, Q. Aplace: A general analytic placement framework. In *Proceedings of the 2005 international symposium on Physical design*, 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*, pp. 84–89, 2004.

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*, 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*, pp. 193–200, 2012.

Kirkpatrick, S., Gelatt Jr, C. D., and Vecchi, M. P. Optimization by simulated annealing. *science*, 220(4598): 671–680, 1983.

Lai, Y., Mu, Y., and Luo, P. Maskplace: Fast chip placement via reinforced visual representation learning. *Advances in Neural Information Processing Systems*, 35:24019–24030, 2022.

Lai, Y., Liu, J., Tang, Z., Wang, B., Hao, J., and Luo, P. Chipformer: Transferable chip placement via offline decision transformer. *arXiv preprint arXiv:2306.14744*, 2023.

Li, X., Zhu, F., Zhen, H.-L., Luo, W., Lu, M., Huang, Y., Fan, Z., Zhou, Z., Kuang, Y., Wang, Z., Geng, Z., Li, Y., Liu, H., An, Z., Yang, M., Li, J., Wang, J., Yan, J., Sun, D., Zhong, T., Zhang, Y., Zeng, J., Yuan, M., Hao, J., Yao, J., and Mao, K. Machine learning insides optverse ai solver: Design principles and applications, 2024.

Liao, P., Liu, S., Chen, Z., Lv, W., Lin, Y., and Yu, B. Dreamplace 4.0: Timing-driven global placement with momentum-based net weighting. In *2022 Design, Automation & Test in Europe Conference & Exhibition (DATE)*, pp. 939–944. IEEE, 2022.

Lin, Y., Dhar, S., Li, W., Ren, H., Khailany, B., and Pan, D. Z. Dreamplace: Deep learning toolkit-enabled gpu acceleration for modern vlsi placement. In *Proceedings of the 56th Annual Design Automation Conference 2019*, pp. 1–6, 2019.

Lin, Y., Pan, D. Z., Ren, H., and Khailany, B. Dreamplace 2.0: Open-source gpu-accelerated global and detailed placement for large-scale vlsi designs. In *2020 China Semiconductor Technology International Conference (CSTIC)*, pp. 1–4. IEEE, 2020.

Ling, H., Wang, Z., and Wang, J. Learning to stop cut generation for efficient mixed-integer linear programming. *Proceedings of the AAAI Conference on Artificial Intelligence*, 38(18):20759–20767, Mar. 2024. doi: 10.1609/aaai.v38i18.30064. URL https://ojs.aaai.org/index.php/AAAI/article/view/30064.

MacMillen, D., Camposano, R., Hill, D., and Williams, T. W. An industrial view of electronic design automation. *IEEE transactions on computer-aided design of integrated circuits and systems*, 19(12):1428–1448, 2000.

Mankowitz, D. J., Michi, A., Zhernov, A., Gelmi, M., Selvi, M., Paduraru, C., Leurent, E., Iqbal, S., Lespiau, J.-B., Ahern, A., et al. Faster sorting algorithms discovered using deep reinforcement learning. *Nature*, 618(7964): 257–263, 2023.

Markov, I. L., Hu, J., and Kim, M.-C. Progress and challenges in vlsi placement research. In *Proceedings of the International Conference on Computer-Aided Design*, pp. 275–282, 2012.

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.

Murata, H., Fujiyoshi, K., Nakatake, S., and Kajitani, Y. Rectangle-packing-based module placement. *Proceedings of IEEE International Conference on Computer Aided Design (ICCAD)*, pp. 472–479, 1995. URL https://api.semanticscholar.org/CorpusID:13916081.

Murata, H., Fujiyoshi, K., Nakatake, S., and Kajitani, Y. Vlsi module placement based on rectangle-packing by the sequence-pair. *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, 15(12):1518–1524, 1996.

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*, pp. 216–220, 2005.

Rabaey, J. M., Chandrakasan, A. P., and Nikolic, B. *Digital integrated circuits*, volume 2. Prentice hall Englewood Cliffs, 2002.

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*, 25(7): 1313–1326, 2006.

Schrittwieser, J., Antonoglou, I., Hubert, T., Simonyan, K., Sifre, L., Schmitt, S., Guez, A., Lockhart, E., Hassabis, D., Graepel, T., et al. Mastering atari, go, chess and shogi by planning with a learned model. *Nature*, 588(7839): 604–609, 2020.

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

Sherwani, N. A. *Algorithms for VLSI physical design automation*. Springer Science & Business Media, 2012.

Shi, Y., Xue, K., Song, L., and Qian, C. Macro placement by wire-mask-guided black-box optimization. In *Thirty-seventh Conference on Neural Information Processing Systems*, 2023.

Shunmugathammal, M., Christopher Columbus, C., and Anand, S. A novel b* tree crossover-based simulated annealing algorithm for combinatorial optimization in vlsi fixed-outline floorplans. *Circuits, Systems, and Signal Processing*, 39:900–918, 2020.

Silver, D., Huang, A., Maddison, C. J., Guez, A., Sifre, L., Van Den Driessche, G., Schrittwieser, J., Antonoglou, I., Panneershelvam, V., Lanctot, M., et al. Mastering the game of go with deep neural networks and tree search. *nature*, 529(7587):484–489, 2016.

Silver, D., Schrittwieser, J., Simonyan, K., Antonoglou, I., Huang, A., Guez, A., Hubert, T., Baker, L., Lai, M., Bolton, A., et al. Mastering the game of go without human knowledge. *nature*, 550(7676):354–359, 2017.

Spindler, P. and Johannes, F. M. Fast and accurate routing demand estimation for efficient routability-driven placement. In *2007 Design, Automation & Test in Europe Conference & Exhibition*, pp. 1–6. IEEE, 2007.

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*, 27(8):1398–1411, 2008.

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.

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*, 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*, pp. 135–140. IEEE, 2007b.

Wang, J., Wang, Z., Li, X., Kuang, Y., Shi, Z., Zhu, F., Yuan, M., Zeng, J., Zhang, Y., and Wu, F. Learning to cut via hierarchical sequence/set model for efficient mixed-integer programming, 2024a.

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

Wang, Z., Wang, J., Zhou, Q., Li, B., and Li, H. Sample-efficient reinforcement learning via conservative model-based actor-critic. *Proceedings of the AAAI Conference on Artificial Intelligence*, 36(8): 8612–8620, Jun. 2022. doi: 10.1609/aaai.v36i8.20839. URL https://ojs.aaai.org/index.php/AAAI/article/view/20839.

Wang, Z., Li, X., Wang, J., Kuang, Y., Yuan, M., Zeng, J., Zhang, Y., and Wu, F. Learning cut selection for mixed-integer linear programming via hierarchical sequence model. In *The Eleventh International Conference on Learning Representations*, 2023a. URL https://openreview.net/forum?id=Zob4P9bRNcK.

Wang, Z., Pan, T., Zhou, Q., and Wang, J. Efficient exploration in resource-restricted reinforcement learning. *Proceedings of the AAAI Conference on Artificial Intelligence*, 37(8):10279–10287, Jun. 2023b. doi: 10.1609/aaai.v37i8.26224. URL https://ojs.aaai.org/index.php/AAAI/article/view/26224.

Wang, Z., Chen, L., Wang, J., Bai, Y., Li, X., Li, X., Yuan, M., Hao, J., Zhang, Y., and Wu, F. A circuit domain generalization framework for efficient logic synthesis in chip design. In *Forty-first International Conference on Machine Learning*. PMLR, 2024b.

Wang, Z., Wang, J., Zuo, D., Ji, Y., Xia, X., Ma, Y., Hao, J., Yuan, M., Zhang, Y., and Wu, F. A hierarchical adaptive multi-task reinforcement learning framework for multiplier circuit design. In *Forty-first International Conference on Machine Learning*. PMLR, 2024c.

Yang, R., Wang, J., Geng, Z., Ye, M., Ji, S., Li, B., and Wu, F. Learning task-relevant representations for generalization via characteristic functions of reward sequence distributions. In *Proceedings of the 28th ACM SIGKDD Conference on Knowledge Discovery and Data Mining*, pp. 2242–2252, 2022.

Ye, W., Liu, S., Kurutach, T., Abbeel, P., and Gao, Y. Mastering atari games with limited data. *Advances in Neural Information Processing Systems*, 34:25476–25488, 2021.

Zhang, Z., Sun, Y., Ye, J., Liu, T.-S., Zhang, J., and Yu, Y. Flow to better: Offline preference-based reinforcement learning via preferred trajectory generation. In *The Twelfth International Conference on Learning Representations*, 2023.

A. Implementation Details

A.1. Model Implementation

HPWL Calculation. We present a detailed example in Figure 7 to illustrate the calculation of HPWL.

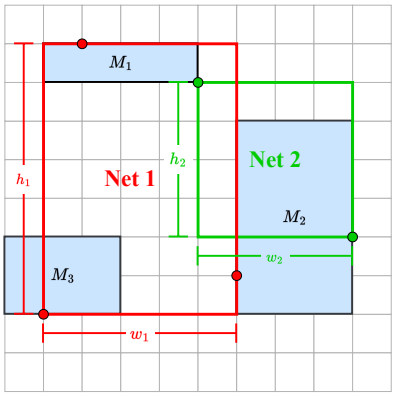


Figure 7. **The calculation of HPWL.** The blue rectangles M_1, M_2 and M_3 represent the placed macros. Solid boxes in red and green represent illustrate the bounding boxes for two distinct nets on the canvas. Pins connected to each net are marked as colored dots, with red dots for pins in Net 1 and green dots for pins in Net 2. Here, HPWL is $w_1 + h_1 + w_2 + h_2 = 20$.

RL Agent. As introduced in Section 4.3, our RL agent employs a U-Net architecture. Below, we provide a comprehensive visualization of our policy network's structure in Figure 8. For a complete understanding of EfficientPlace's configuration, the specific hyperparameters are listed in Table 2.

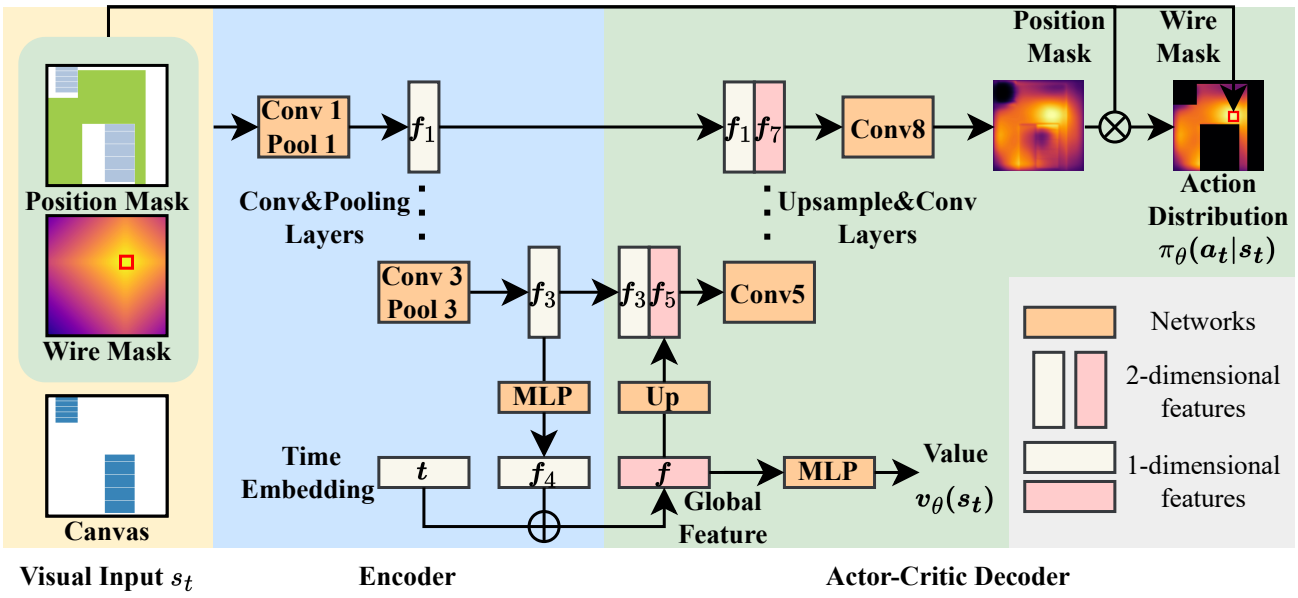


Figure 8. **RL Agent Architecture.** Our policy network adopts a U-Net architecture. The encoding stage, on the U-Net's left side, processes visual inputs, incorporating netlist information and timestep embeddings to enhance contextual awareness. Conversely, the decoder on the right side generates policy probabilities. Features from the encoder are integrated into the decoder to ensure the full transmission of input data, enabling precise control over placement decisions. The position mask and wiremask are then used to limit the output actions, thus effectively reducing the vast search space.

Table 2. **Model Hyperparameters.**

	layer name	kernel size	output size
Actor	CNN_1	3×3, 8	(8,256,256)
	CNN_2	3×3, 16	(16,128,128)
	CNN_3	3×3, 32	(32,16,16)
	CNN_4	3×3, 32	(32,4,4)
	FC	-	(512,)
Critic	CNN_1	3×3, 8	(8,256,256)
	CNN_2	3×3, 16	(16,128,128)
	CNN_3	3×3, 32	(32,16,16)
	CNN_4	3×3, 32	(32,4,4)
	FC_1	-	(512,)
	FC_2	-	(1,)
time_embedding	embedding	-	(32,)

A.2. Proof to Theorem 1

Theorem 1. *Suppose we are at the k^{th} iteration, with the set of frontiers $\mathcal{F}_k$ where $|\mathcal{F}_k| = K_{\mathcal{F}}$, the RL policy π_k, and the evaluation Q_k. After conducting multiple RL rollouts, we obtain an improved policy π_{k+1}. We assume that every $s \in \mathcal{F}_k$ is visited, and that $v^{\pi_{k+1}}(s) \geq v^{\pi_k}(s)$ for every $s \in \mathcal{F}_k$, where v^{π} represents the value function. This iteration results in the updated Q_{k+1} and $\mathcal{F}_{k+1}$. Then we have*

$$\mathbb{E}_{s\sim\mathcal{F}_{k+1}}[Q_{k+1}(s)] \geq \mathbb{E}_{s\sim\mathcal{F}_k}[Q_k(s)], \tag{1}$$

and

$$\mathbb{E}_{\pi_{k+1}}[\mathbb{E}_{s\sim\mathcal{F}_{k+1}}[Q_{k+1}(s)]] \geq \mathbb{E}_{s\sim\mathcal{F}_k}[v^{\pi_{k+1}}(s)]. \tag{2}$$

Proof. According to the updating rule in Algorithm 2, a frontier candidate s is updated as a frontier node only when it replaces another frontier node s' with lower Q value. That is, for $s \in \mathcal{F}_{k+1} \setminus \mathcal{F}_k$, there exists a corresponding $s' \in \mathcal{F}_k \setminus \mathcal{F}_{k+1}$, such that $Q_{k+1}(s') \leq Q_{k+1}(s)$. Then have

$$\begin{aligned}
\mathbb{E}_{s\sim\mathcal{F}_{k+1}}[Q_{k+1}(s)] =& \frac{1}{|K_{\mathcal{F}}|}\left(\sum_{s\in\mathcal{F}_{k+1}\cap\mathcal{F}_k} Q_{k+1}(s) + \sum_{s\in\mathcal{F}_{k+1}\setminus\mathcal{F}_k} Q_{k+1}(s)\right) \\
\geq& \frac{1}{|K_{\mathcal{F}}|}\left(\sum_{s\in\mathcal{F}_{k+1}\cap\mathcal{F}_k} Q_{k+1}(s) + \sum_{s'\in\mathcal{F}_k\setminus\mathcal{F}_{k+1}} Q_{k+1}(s')\right) \\
=& \mathbb{E}_{s\sim\mathcal{F}_k}[Q_{k+1}(s)].
\end{aligned} \tag{3}$$

By definition, we have

$$Q_k(s) = \max_{s_T\in\mathcal{P}_k(s)} -\text{HPWL}(s_T), \tag{4}$$

where $\mathcal{P}_k$ denotes the set of descendant leaf nodes of s in the k^{th} search tree. As the tree expands, we have $\mathcal{P}_k(s) \subset \mathcal{P}_{k+1}(s)$, which derives $Q_{k+1}(s) \geq Q_k(s)$. Thus we have

$$\mathbb{E}_{s\sim\mathcal{F}_{k+1}}[Q_{k+1}(s)] \geq \mathbb{E}_{s\sim\mathcal{F}_k}[Q_{k+1}(s)] \geq \mathbb{E}_{s\sim\mathcal{F}_k}[Q_k(s)]. \tag{5}$$

Since each node $s \in \mathcal{F}_k$ is assumed to be visited, the current $Q_{k+1}(s)$ is an evaluation of the current policy π_{k+1}. Thus we have $\mathbb{E}_{\pi_{k+1}}Q_{k+1}(s) \geq v_{\pi_{k+1}}(s)$, which derives

$$\mathbb{E}_{\pi_{k+1}}[\mathbb{E}_{s\sim\mathcal{F}_{k+1}}[Q_{k+1}(s)]] \geq \mathbb{E}_{s\sim\mathcal{F}_k}[\mathbb{E}_{\pi_{k+1}}[Q_{k+1}(s)]] \geq \mathbb{E}_{s\sim\mathcal{F}_k}[v^{\pi_{k+1}}(s)]. \tag{6}$$

□

A.3. Dataset & Experimental Details

Table 3 details the statistics for eight circuits from the `ISPD2005` benchmark, used as our test datasets. The "Macros (to place)" column specifies the quantity of macros chosen for placement in our study. For *bigblue2* and *bigblue4*, due to their extensive numbers of of macros, we follow Lai et al. (2023) and select 256 and 1024 macros, respectively, for a fair comparison.

Table 3. **Statistics of public benchmark circuits.**

Circuit	Macros	Macros (to place)	Hard Macros	Standard Cells	Nets	Pins	Area Util(%)
adaptec1	543	543	63	210904	221142	944063	55.62
adaptec2	566	566	159	254457	266009	1069482	74.46
adaptec3	723	723	201	450927	466758	1875039	61.51
adaptec4	1329	1329	92	494716	515951	1912420	48.62
bigbule1	560	560	32	277604	284479	1144691	31.58
bigbule2	23084	256	52	534782	577235	2122282	32.43
bigbule3	1293	1293	138	1095519	1123170	3833218	66.81
bigbule4	8170	1024	52	2169183	2229886	8900078	35.68

Experimental hyperparameters, including those for macro and subsequent standard cell placement, are outlined in Table 4. "Num of update epochs" refers to the epochs count utilized per RL update, which is set as 10. "Update frontiers begin" denotes the training phase warm up for frontier updates, which is set as 200. "Update frontiers freq" establishes how often frontiers are updated, which is set as 2. "Num of episodes" indicates the episode count per PPO updating loop, which is set as 5. We employ different grid sizes for different circuits according to their characteristics, opting for resolutions as 128 for *adaptec1,2* and *bigblue2*, 256 for *bigblue3,4*, and 512 for *adaptec3,4*.

Table 4. **Hyper-parameters used in our experiments.**

Configuration	Value
num of update epochs	10
update frontiers freq	2
update frontiers begin	200
num of episodes	5
density weight in (phase 2)	0
number of iterations (phase 2)	1000
density weight in (phase 3)	8e-3

B. Additional Results

B.1. Time Efficiency

Figure 9 presents the HPWL performance of EfficientPlace and BBO vs. wall clock time, further demonstrating the advantage in efficiency of our approach.

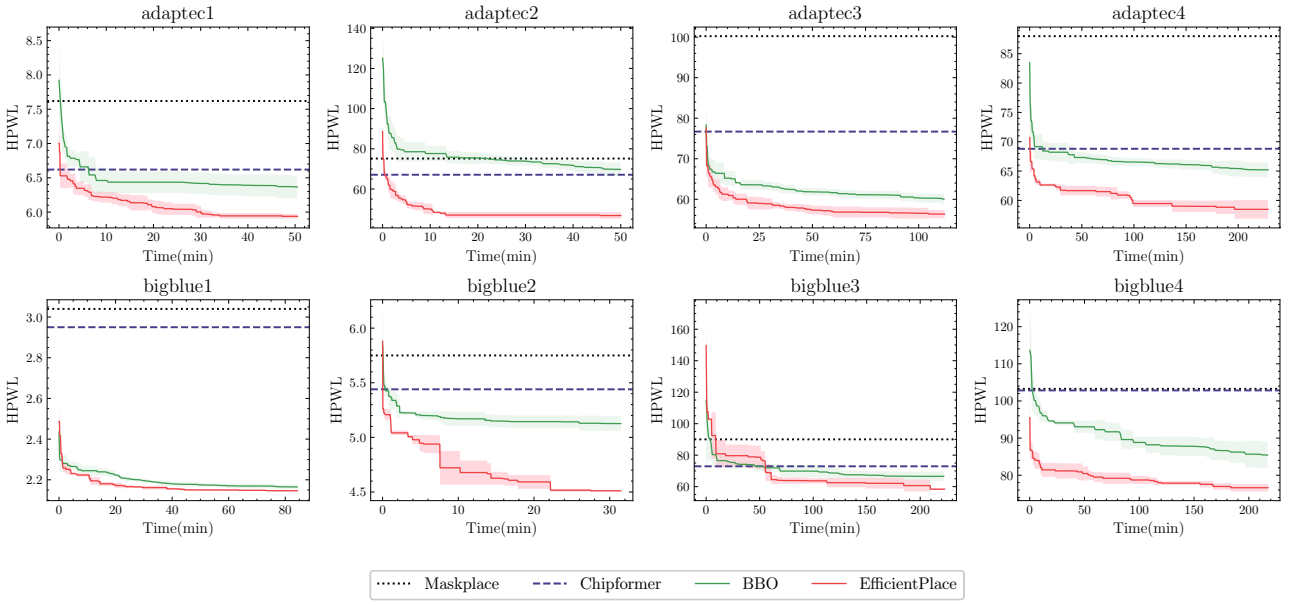


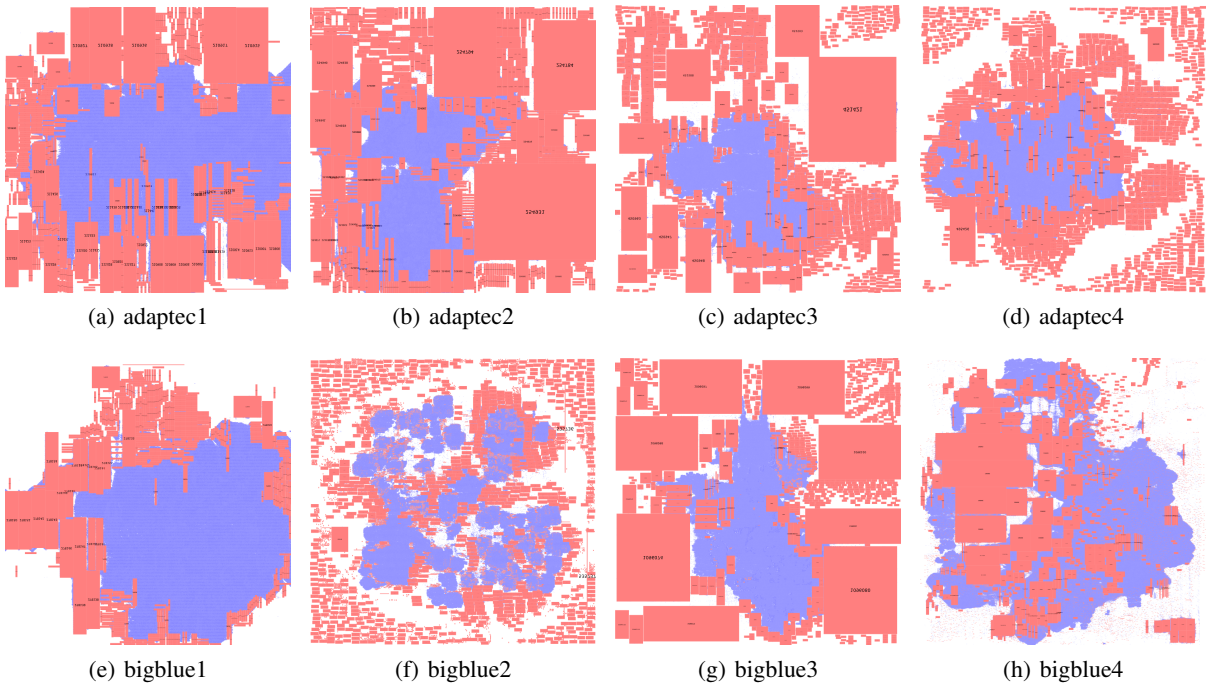
Figure 9. **Results of Runtime.** HPWL curve on runtime of EfficientPlace compared with BBO. EfficientPlace's HPWL curve consistently remains below that of BBO within the runtime range tested, while also achieving faster convergence.

B.2. Mixed-Size Placement Results

Table 5 presents the results of mixed-size placement HPWL metrics across various methods. The results illustrate EfficientPlace's capability to excel in mixed-size placement. The benchmark settings and results for other methodologies, including DREAMPLACE (Liao et al., 2022), SP-SA (Murata et al., 1995), MaskPlace (Lai et al., 2022), and WireMask-EA (Shi et al., 2023), are from Shi et al. (2023). EfficientPlace approaches the mixed-size placement challenge in two distinct phases: (1) initial cell placement using DREAMPlace (Liao et al., 2022) based on the macro positions determined by our method, bypassing the legalization phase; followed by (2) a comprehensive mixed placement phase utilizing DREAMPlace, which integrates the initial placement with further legalization and detailed placement processes. Since WireMask-EA (Shi et al., 2023) does not include *bigblue2* in their experiments, we extend our evaluation to this dataset using their prescribed settings for a thorough comparison. Furthermore, Figure 10 offers visual representations of placements achieved by EfficientPlace for all circuits within the `ISPD2005` benchmark, demonstrating our method's effectiveness visually.

Table 5. **Mixed-size Placed Results.** Comparison of HPWL values ($\times 10^7$) of all methods listed on mixed-size placement task. The best results (the lowest HPWL) are highlighted in bold. EfficientPlace can also achieve the best mixed-size placement result.

benchmark	adaptec1	adaptec2	adaptec3	adaptec4	bigbule1	bigbule2	bigbule3	bigbule4
DREAMPlace	11.10 ±1.31	13.84 ± 1.74	17.03 ± 0.99	24.37 ± 1.13	10.06 ± 0.28	/	36.51 ± 0.56	175.86 ± 2.23
SP-SA+DREAMPlace	10.18 ± 0.18	14.80 ± 0.01	30.63 ± 0.82	30.63 ± 0.82	10.70 ± 0.01	/	63.60 ± 0.12	203.79 ± 0.36
MaskPlace+DREAMPlace	10.86 ± 0.01	12.98 ± 0.58	26.14 ± 0.07	26.14 ± 0.07	10.64 ± 0.01	/	54.98 ± 1.06	/
WireMask-EA+DREAMPlace	8.93 ± 0.01	**9.20 ± 0.05**	21.72 ± 0.01	21.72 ± 0.01	10.35 ± 0.02	14.88±0.01	42.52 ± 0.11	171.23 ± 0.48
EfficientPlace(ours)	**7.20±0.12**	**9.20±0.61**	**16.49±1.07**	**14.70±0.25**	**8.67±0.10**	**9.98±0.02**	**28.48±0.96**	**125.02±0.02**



(a) adaptec1 (b) adaptec2 (c) adaptec3 (d) adaptec4

(e) bigblue1 (f) bigblue2 (g) bigblue3 (h) bigblue4

Figure 10. **Visualization of mixed-size placement results by EfficientPlac for ISPD2005 benchmark.** Macros are marked in red, while standard cells are represented in blue.

B.3. Congestion Results

Table 6 presents a comparison of congestion and HPWL metrics between EfficientPlace and the state-of-the-art methods MaskPlace (Lai et al., 2022) and WireMask-EA (Shi et al., 2023), utilizing RUDY for congestion calculations. In alignment with Shi et al. (2023), we standardize our congestion values to 1 for a direct comparison. Given the impact of grid size on RUDY's outcomes, we ensure a consistent canvas partitioning across all methods for each evaluation. In both congestion and HPWL assessments, EfficientPlace demonstrates superior performance.

Table 6. **Congestion Results.** Comparison on HPWL ($\times 10^5$) and Congestion (Cong., lower are better)

Benchmark	adaptec1		adaptec2		adaptec3		adaptec4		bigblue1		bigblue2		bigblue3		bigblue4	
Metrics	HPWL	Cong.	HPWL	Cong.	HPWL	Cong.	HPWL	Cong.	HPWL	Cong.	HPWL	Cong.	HPWL	Cong.	HPWL	Cong.
MaskPlace	7.10	1.17	85.46	2.09	84.91	1.18	75.46	1.20	2.6	4.43	/	/	114.01	1.70	/	/
EA	5.97	1.21	61.01	1.46	59.03	1.11	63.22	1.04	2.17	1.45	4.98	1.01	61.39	1.10	90.7	1.04
OURS	**5.9**	**1**	**45.3**	**1**	**55.56**	**1**	**56.35**	**1**	**2.07**	**1**	**4.65**	**1**	**55.28**	**1**	**75.8**	**1**