Energy-Guided Continuous Diffusion for Unsupervised Chip Placement

Anonymous Author(s)

Abstract

Chip placement is a crucial process in integrated circuit design. The quality of the placement directly impacts performance, power consumption, and routability. Recent machine learning-based approaches have shown promising results but they also face significant limitations. Reinforcement learning (RL) methods suffer from poor generalization and require extensive retraining for each new circuit. Existing diffusion model-based methods are generalizable but lack explicit optimization objectives during the generation process. This article presents an energy-guided continuous diffusion model trained by using policy gradients. This method starts from random placements and iteratively refines them through a denoising process. This denoising process is guided by an energy function that represents the placement quality. The energy-guided diffusion approach not only enables parallel placement of all components but also requires no optimal placement examples for training. It enables efficient training on diverse synthetic data that covers most of the real chip data distributions. Therefore, the model learns general placement strategies that can be transferred to new designs without retraining. Experiments on standard benchmarks (ISPD 2005, ICCAD 2004) demonstrate that the proposed method achieves state-of-the-art placement quality with superior cross-circuit generalization.

Keywords

Chip placement, diffusion models, reinforcement learning, proximal policy optimization, electronic design automation, denoising diffusion, macro placement, physical design

ACM Reference Format:
Anonymous Author(s). 2026. Energy-Guided Continuous Diffusion for Unsupervised Chip Placement. In *Proceedings of Design Automation Conference (DAC'26)*. ACM, New York, NY, USA, 7 pages. https://doi.org/XXXXXXX.XXXXXXX

1 Introduction

Chip placement is the process of determining the optimal physical locations of circuit components on a two-dimensional silicon canvas. The quality of the placement directly affects key design metrics such as wirelength, routability, power consumption, and timing performance. This process remains one of the most critical and computationally intensive steps in modern integrated circuit design [13, 15].

DAC'26, Long Beach, CA, USA
© 2026 Copyright held by the owner/author(s). Publication rights licensed to ACM.
ACM ISBN 978-1-4503-XXXX-X/26/07
https://doi.org/XXXXXXX.XXXXXXX

Traditional analytical methods formulate placement as continuous optimization [13], but it often converge to local optima due to the non-convex optimization landscape [25, 26]. Machine learning-based approaches have shown promising results but face fundamental limitations: Reinforcement learning (RL) placement methods, such as CircuitTraining [14], exhibit poor generalization and require expensive retraining for each new circuit [6, 27]. In addition, by casting placement as a Markov Decision Process, sequential RL-based placements face challenges that the suboptimal placements during the early stage of the generation cannot be reverted [12]. Recent diffusion-based approaches [12] demonstrate improved generalization through zero-shot transfer to new circuits that never appeared in the training data. However, they rely on supervised learning from optimal placement examples, which are expensive to acquire through human experts or commercial Electronic Design Automation (EDA) tools. Additionally, existing diffusion-based methods apply guidance only during inference rather than training, limiting the model's ability to learn explicit optimization strategies [12].

We propose a continuous diffusion framework that addresses these limitations by leveraging energy-based guidance during training itself. This approach extends Scalable Discrete Diffusion Samplers (SDDS) [17] from discrete combinatorial optimization to continuous chip placement. Unlike sequential RL methods that place components one at a time, the proposed diffusion approach performs parallel optimization over all components. Critically, energy is computed at every diffusion step rather than only at the final placement, providing dense feedback signals that significantly improve sample efficiency beyond standard policy-gradient diffusion methods. This approach enables unsupervised learning directly from the energy function without requiring optimal placement examples, allowing efficient training on diverse synthetic data. The main contributions are:

- We extend policy gradient-based diffusion training from discrete to continuous domains for chip placement, enabling unsupervised learning from energy functions without optimal placement examples. Unlike supervised diffusion methods that require costly ground-truth placements, this approach learns directly from placement quality metrics.
- We introduce dense per-time step energy feedback that provides immediate gradient signals rather than propagating sparse terminal rewards across long trajectories. This reduces gradient variance by 3.2× compared to sparse-reward baselines.
- We demonstrate zero-shot generalization from synthetic training data to real benchmarks (ISPD 2005, ICCAD 2004), achieving 3.7–5.6% Half-perimeter wire length (HPWL) improvements over state-of-the-art supervised diffusion baselines without per-circuit retraining.

The remainder of this paper is organized as follows. Section 2 reviews related work. Section 3 introduces the preliminaries. Section 4 presents the energy-guided continuous diffusion framework. Section 5 presents experimental results. Section 6 concludes the paper and discusses future directions. The source code can be found here[1]

2 Related Work

Analytical Methods. Traditional chip placement methods like DREAMPlace [13] formulate the problem as continuous optimization with GPU-accelerated differentiable engines. While achieving strong performance, they often converge to local optima due to the highly non-convex placement landscape [25, 26] and require careful manual tuning that may not generalize well across designs.

Reinforcement Learning Methods. Google's CircuitTraining [14] pioneered deep RL for chip placement but suffers from poor generalization requiring expensive per-circuit retraining [6], irreversible sequential placement decisions [12], and sparse reward credit assignment [11]. While MaskPlace [11], ChiPFormer [10], and EfficientPlace [3] improve training efficiency, cross-circuit generalization remains unresolved.

Diffusion Models. Lee et al. [12] demonstrated that diffusion models achieve competitive chip placement through zero-shot transfer by training on synthetic data using supervised learning from near-optimal placements, then applying guided sampling during inference.

RL for Diffusion. Recent work trains diffusion models with reinforcement learning, enabling optimization without optimal examples. Black et al. [1] fine-tuned text-to-image models via policy gradients. For combinatorial optimization, DiffUCO [18] and SDDS [17] used energy-based guidance with PPO for discrete problems. Our work extends RL-based diffusion training to continuous chip placement, learning directly from placement quality metrics.

3 Preliminaries

3.1 Diffusion Models

Continuous diffusion models generate data by learning to reverse a gradual noising process [4, 22]. The forward diffusion process progressively corrupts data $\mathbf{x}_0$ by adding Gaussian noise over T time steps. Starting from clean data at $t = 0$, increasingly noisy states are obtained through:

$$\mathbf{x}_t = \sqrt{1-\beta_t} \cdot \mathbf{x}_{t-1} + \sqrt{\beta_t} \cdot \boldsymbol{\epsilon}, \quad \boldsymbol{\epsilon} \sim \mathcal{N}(0, \mathbf{I}) \tag{1}$$

where $\beta_t \in (0, 1)$ defines the noise schedule. A cosine schedule [16] is commonly used to control noise injection across time steps. At $t = T$, the data is corrupted to pure Gaussian noise.

Then the reverse process iteratively denoises from random noise back to clean data. In this denoising process, a neural network q_θ learns to predict the conditional distribution:

$$q_\theta(\mathbf{x}_{t-1}|\mathbf{x}_t, t) = \mathcal{N}(\boldsymbol{\mu}_\theta(\mathbf{x}_t, t), \boldsymbol{\sigma}^2_\theta(\mathbf{x}_t, t) \cdot \mathbf{I}) \tag{2}$$

This framework has achieved remarkable success in image generation [2], audio synthesis [9], video generation [5], and molecular conformation [24], demonstrating its versatility across diverse continuous domains.

[1] https://anonymous.4open.science/r/UCP

3.2 Policy Gradient Training

Traditional diffusion models are trained to match data distributions via maximum likelihood [4] or score matching [22]. Although they enable high-quality result generations, these approaches typically require high volumes of high-quality examples from the target distribution. For chip placement, optimal placements require expensive commercial tools or human expertise, which is difficult to acquire or scale.

An alternative approach is to train diffusion models through *policy gradients* from reinforcement learning (RL) [23], optimizing directly for solution quality without requiring optimal examples. This method treats the reverse diffusion process as a decision-making policy that iteratively refines the entire placement through denoising steps. In this view:

- The diffusion model $q_\theta(\mathbf{x}_{t-1}|\mathbf{x}_t, t)$ acts as a policy that samples the next state $\mathbf{x}_{t-1}$ given current state $\mathbf{x}_t$.
- A complete diffusion trajectory $\mathbf{x}_{T:0} = (\mathbf{x}_T, \mathbf{x}_{T-1}, \ldots, \mathbf{x}_0)$ from noise to solution corresponds to an episode.
- The final state $\mathbf{x}_0$ is evaluated by a reward function $R(\mathbf{x}_0)$ measuring solution quality.

The policy gradient theorem [23] provides a way to compute gradients of the expected reward with respect to policy parameters θ:

$$\nabla_\theta \mathbb{E}_{\mathbf{x}_{T:0}\sim q_\theta}[R(\mathbf{x}_0)] = \mathbb{E}_{\mathbf{x}_{T:0}\sim q_\theta}\left[\sum_{t=1}^{T} \nabla_\theta \log q_\theta(\mathbf{x}_{t-1}|\mathbf{x}_t, t) \cdot \hat{A}_t\right] \tag{3}$$

where $\hat{A}_t$ is the advantage function quantifying how much better the sampled action at time step t is compared to the expected value. This gradient can be estimated from sampled trajectories, enabling learning without supervised labels.

In practice, policy gradient optimization is performed using PPO [20], an on-policy RL algorithm that stabilizes training by limiting policy updates through a clipped objective. PPO uses Generalized Advantage Estimation (GAE) [19] to compute advantages $\hat{A}_t$ from temporal differences along the trajectory. This combination provides stable, sample-efficient learning for stochastic policies.

SDDS [17] applies this policy gradient framework to train diffusion models for discrete combinatorial optimization problems such as the Traveling Salesman Problem (TSP) and Maximum Independent Set (MIS). SDDS operates on binary state spaces $\{0, 1\}^N$ and it uses a sparse reward structure. It means that the reward function evaluates only the final solution $R(\mathbf{x}_0)$, providing no intermediate feedback during the diffusion trajectory.

This sparse reward structure leads to the credit assignment problem [23]. This is a fundamental challenge in RL because it is difficult to determine which of the T sequential decisions contributed to the final outcome. In the case of diffusion models, this single reward must be distributed backward across the entire trajectory through GAE's exponential decay mechanism $(\gamma\lambda)^l$. For structured problems like chip placement, where components interact through long-range nets spanning the circuit, this creates high-variance gradient estimates as the optimization signal becomes exponentially attenuated for early time steps [23].

4 Methodology

4.1 Problem Formulation

Chip placement is formulated as an energy minimization problem over continuous 2D coordinates. Given a netlist $\mathcal{G} = (\mathcal{V}, \mathcal{E})$ with N components (macros and standard cells) and nets connecting them, the goal is to find positions $\mathbf{x} = \{(x_i, y_i)\}_{i=1}^{N}$ on a 2D canvas of dimensions $W \times H$ that minimize an energy function $E(\mathbf{x})$ while satisfying placement constraints. Following standard practice [12], all component positions and sizes are normalized to a $[-1, 1] \times [-1, 1]$ coordinate system (canvas center $c = (0, 0)$, half-width $L = 1$ in each dimension). This normalization enables training on diverse circuit sizes and simplifies the energy formulation.

To evaluate placement quality, two key metrics are used, following [12]. First, the legality measures whether a placement satisfies physical constraints: no component overlaps, all components within canvas boundaries. Legality is defined as the ratio A_u/A_s where A_u is the area within the circuit boundary occupied by the union of all placed components, and A_s is the sum of individual component areas. A legality score of 1 indicates that all constraints are satisfied. Second, HPWL serves as a proxy for chip performance by approximating routed wirelength. Our objective is therefore to generate legal placements ($A_u/A_s = 1$) with minimal HPWL.

The goal is to learn a conditional distribution $q_\theta(\mathbf{x}|\mathcal{G})$ that maps netlists to high-quality placements by minimizing expected energy:

$$\theta^* = \arg\min_{\theta} \mathbb{E}_{\mathcal{G}\sim p(\mathcal{G})} \mathbb{E}_{\mathbf{x}\sim q_\theta(\cdot|\mathcal{G})} [E(\mathbf{x})] \tag{4}$$

The energy function captures placement quality through three differentiable components:

$$E(\mathbf{x}) = E_{\text{wire}}(\mathbf{x}) + \lambda_{\text{overlap}} E_{\text{overlap}}(\mathbf{x}) + \lambda_{\text{bound}} E_{\text{bound}}(\mathbf{x}) \tag{5}$$

Following ChipDiffusion [12], the wirelength term computes HPWL for each net $e \in \mathcal{E}$:

$$E_{\text{wire}}(\mathbf{x}) = \sum_{e\in\mathcal{E}} \left[\max_{i\in e}(x_i) - \min_{i\in e}(x_i) + \max_{i\in e}(y_i) - \min_{i\in e}(y_i) \right] \tag{6}$$

The overlap penalty penalizes pairwise component overlaps following ChipDiffusion [12]:

$$E_{\text{overlap}}(\mathbf{x}) = \sum_{i<j} \left[\min(0, d_{ij}(\mathbf{x})) \right]^2 \tag{7}$$

where $d_{ij}(\mathbf{x})$ is the signed distance between rectangular components i and j. For axis-aligned rectangles with centers (x_i, y_i), (x_j, y_j) and half-sizes $(w_i/2, h_i/2)$, $(w_j/2, h_j/2)$:

$$d_{ij}(\mathbf{x}) = \max\left(|x_i - x_j| - \frac{w_i + w_j}{2}, |y_i - y_j| - \frac{h_i + h_j}{2} \right) \tag{8}$$

When components overlap, $d_{ij} < 0$ and the penalty is activated. When components do not overlap, $d_{ij} > 0$ and $\min(0, d_{ij}) = 0$, resulting in zero penalty. The squaring makes the penalty differentiable everywhere and emphasizes larger violations.

The boundary penalty enforces canvas constraints by penalizing components that extend outside the $[-1, 1] \times [-1, 1]$ canvas:

$$E_{\text{bound}}(\mathbf{x}) = \sum_{i=1}^{N} \left[\max\left(0, |x_i| + \frac{w_i}{2} - 1\right)^2 + \max\left(0, |y_i| + \frac{h_i}{2} - 1\right)^2 \right] \tag{9}$$

For each component i, we compute how far each edge extends beyond the canvas boundary. The max function returns zero if the component is within bounds, and the violation distance otherwise. The squared penalty provides smooth gradients for optimization.

4.2 Training Method

We extend the SDDS policy gradient framework in two key ways. First, we adapt it from discrete binary state spaces $\{0, 1\}^N$ to continuous 2D chip placement coordinates $\mathbf{x} \in \mathbb{R}^{2N}$. Second, we address the credit assignment problem by introducing dense energy feedback. The energy is computed at every diffusion time step rather than only after the final solution.

The policy gradient framework from Equation 3 and PPO algorithm apply directly to this continuous setting. However, unlike SDDS and similar policy-gradient methods [14, 17] that use sparse terminal rewards $R(\mathbf{x}_0)$, we define per-step rewards that include the energy evaluated at the current state. For a trajectory $\mathbf{x}_{T:0} = (\mathbf{x}_T, \mathbf{x}_{T-1}, \ldots, \mathbf{x}_0)$ sampled from the reverse policy q_θ, we define the per-step reward:

$$r_t(\mathbf{x}_t, \mathbf{x}_{t-1}) = -E(\mathbf{x}_t) + \alpha_{\text{noise}} \log p(\mathbf{x}_t|\mathbf{x}_{t-1}) + \alpha_{\text{ent}} \mathcal{H}(q_\theta(\mathbf{x}_{t-1}|\mathbf{x}_t)) \tag{10}$$

where $-E(\mathbf{x}_t)$ provides immediate feedback about placement quality at each step, $\log p(\mathbf{x}_t|\mathbf{x}_{t-1})$ evaluates the reverse trajectory under the forward Gaussian kernel (from Equation 1) to prevent excessive deviation, and $\mathcal{H}(\cdot)$ maintains policy exploration. The noise term does not require sampling from the forward process. It simply evaluates the log-probability of the reverse-sampled transition $(\mathbf{x}_{t-1}, \mathbf{x}_t)$ under the known forward Gaussian distribution.

We optimize the policy using PPO, maximizing expected cumulative reward:

$$\mathcal{J}(\theta) = \mathbb{E}_{\mathbf{x}_{T:0}\sim q_\theta} \left[\sum_{t=1}^{T} r_t(\mathbf{x}_t, \mathbf{x}_{t-1}) \right] \tag{11}$$

GAE computes advantages $\hat{A}_t$ from temporal differences:

$$\delta_t = r_t + \gamma V_\theta(\mathbf{x}_{t+1}) - V_\theta(\mathbf{x}_t), \quad \hat{A}_t = \sum_{l=0}^{T-t} (\gamma\lambda)^l \delta_{t+l} \tag{12}$$

where V_θ is a learned value function and $\hat{A}_t$ is the advantage estimate. Since our per-step reward r_t contains the energy term $-E(\mathbf{x}_t)$ directly, each temporal difference $\delta_t = r_t + \gamma V_\theta(\mathbf{x}_{t+1}) - V_\theta(\mathbf{x}_t)$ incorporates immediate placement quality feedback rather than relying solely on value function estimates. Figure 1 demonstrates a policy that reduces placement energy throughout the diffusion trajectory, which enables progressive refinement from noise to structured placements.

With these two extensions, the model learns a conditional distribution $q_\theta(\mathbf{x}|\mathcal{G})$ through reverse diffusion conditioned on the netlist graph $\mathcal{G}$. Starting from random Gaussian noise at $t = T$, the model predicts Gaussian parameters at each step:

$$q_\theta(\mathbf{x}_{t-1}|\mathbf{x}_t, \mathcal{G}, t) = \mathcal{N}(\boldsymbol{\mu}_\theta(\mathbf{x}_t, \mathcal{G}, t), \boldsymbol{\sigma}_\theta^2(\mathbf{x}_t, \mathcal{G}, t) \cdot \mathbf{I}) \tag{13}$$

4.3 Architecture

The neural network follows an Encode-Process-Decode paradigm with graph neural network (GNN) message passing [18], as illustrated in Figure 2. Node features include component sizes (w_i, h_i), current noisy positions $\mathbf{x}_t = (x_t, y_t)$, and sinusoidal time step t

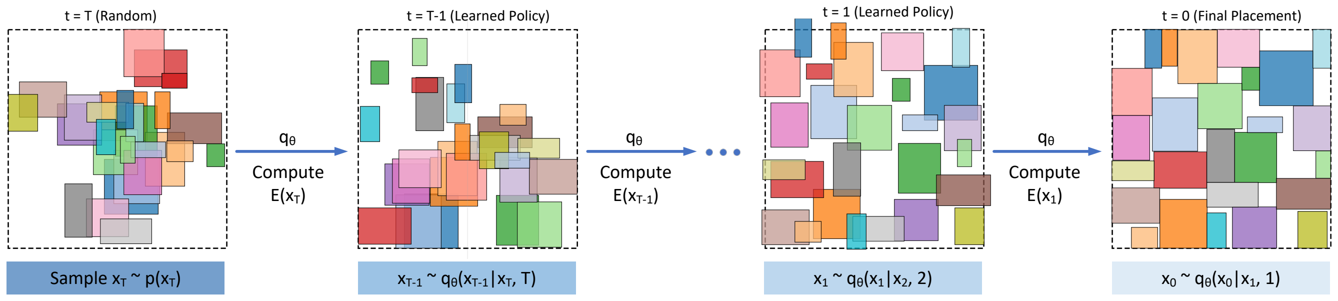

Figure 1: The energy-guided reverse diffusion process: the learned policy q_θ progressively refines component positions from random noise ($t = T$) through intermediate states to final legal placement ($t = 0$).

encodings that inform the model of the current diffusion step. Edge features encode net connectivity from the netlist $\mathcal{G}$.

The processor applies 16 GNN message-passing layers with 256 hidden dimensions, enabling information propagation across the circuit topology. Each layer aggregates neighbor information to capture both local neighborhoods and global circuit structure. The decoder outputs mean positions $\boldsymbol{\mu}_\theta$ in normalized coordinates and log-variances $\log \boldsymbol{\sigma}_\theta^2$ for each component. Following standard diffusion model practices [4], the log-variances are clipped to prevent numerical instability during training. The predicted mean positions are scaled from normalized coordinates to canvas dimensions $W \times H$ during inference.

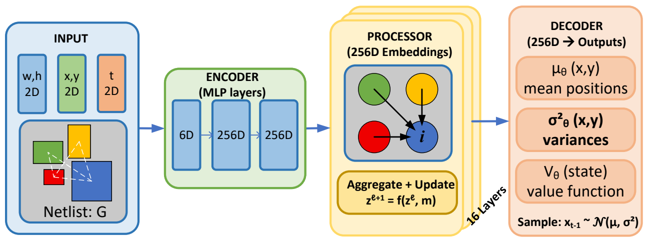

Figure 2: The model architecture used

4.4 Synthetic Training Data Generation

Because our method optimizes the energy function directly through policy gradients, we can generate training data efficiently without constructing near-optimal placements. We obtain synthetic datasets consisting of tuples $(\mathcal{G}, \mathbf{x}_{\text{init}})$ where $\mathcal{G}$ is a netlist and $\mathbf{x}_{\text{init}}$ represents randomized component positions.

We randomly generate components by sampling sizes from clipped exponential distributions. For each component i, the width and height are independently drawn from:

$$s_i \sim \text{ClippedExp}(\lambda_{\text{size}}, s_{\min}, s_{\max}), \quad \lambda_{\text{size}} \sim \text{Uniform}(0.04, 0.08) \tag{14}$$

where $s_{\min} = 0.01$ and $s_{\max} = 1.0$. This distribution captures the characteristic heavy-tailed size distribution of real circuits, where most components are small standard cells with occasional large macros [12]. The total component area satisfies a density constraint:

$$\sum_{i=1}^{N} w_i h_i \le \rho \cdot A_{\text{canvas}}, \quad \rho \sim \text{Uniform}(0.75, 0.9) \tag{15}$$

where $A_{\text{canvas}} = 4$ (for the $[-1,1] \times [-1,1]$ normalized canvas), ensuring sufficient space for legal placements.

To generate netlists with realistic structure and to make sure each training instance does have a legal solution, we first create a temporary legal placement $\mathbf{x}_{\text{temp}}$ using collision detection, then generate three types of connections: (1) **Local proximity (60%):** each component connects to $k_{\text{local}} \sim \text{Uniform}\{2, 4\}$ nearest neighbors via $\mathcal{N}_{\text{local}}(i) = \{j : j \in \text{TopK}(\{d_{ij}\}_{j \ne i})\}$ where $d_{ij} = \|\mathbf{x}_i^{\text{temp}} - \mathbf{x}_j^{\text{temp}}\|_1$. (2) **Hierarchical clusters (30%):** spatial k-means partitions components into clusters ($m \sim \text{Uniform}(8, 16)$ per cluster), with inter-cluster connections weighted by $\exp(-d_{\text{cluster}}/0.5)$. (3) **Long-range (10%):** random component pairs simulate critical paths. We merge 30% of 2-pin nets into multi-pin nets (60% 3-pin, 30% 4-pin, 10% 5-pin).

After netlist construction, component positions are randomized to create training data:

$$\mathbf{x}_{\text{init}} = \{(x_i, y_i)\}_{i=1}^{N}, \quad x_i, y_i \sim \text{Uniform}(-1, 1) \tag{16}$$

This destroys the spatial relationships from the temporary legal placement, creating challenging starting configurations with component overlaps and poor wirelength. During training, the model must learn to resolve these violations and optimize placement quality through policy gradient updates guided by the energy function.

We generate diverse training circuits at realistic scales: component counts $N \sim \text{Uniform}(200, 1000)$, clipped exponential size parameters $\lambda_{\text{size}} \in [0.04, 0.08]$ with bounds $[s_{\min}, s_{\max}] \in [0.01, 1.0]$, multi-scale connectivity (local + hierarchical + long-range), multi-pin nets, and canvas aspect ratios sampled from Uniform$(0.5, 2.0)$.

4.5 Legalization Decoder

The trained diffusion model generates placements through reverse sampling. It produces continuous component positions that may still contain overlaps or boundary violations even with the legality penalty terms in the energy function defined in Equation 5. To guarantee a legal placement, we employ a force-directed legalization algorithm with spatial hashing.

The spatial hashing approach partitions the canvas into a uniform grid, allowing each component to check overlaps only with components in the same or neighboring cells. The force-directed formulation treats overlaps as repulsive forces proportional to penetration depth, while boundary violations are resolved via gradient descent on the boundary penalty E_{bound}. The adaptive step size η

Algorithm 1 Force-Directed Legalization

```
1:  Initialize uniform grid with cell size $\delta = 2 \cdot \mathbb{E}[\max(w_i, h_i)]$
2:  while $A_u/A_s < 0.99$ and iterations $< 1000$ do
3:      Build spatial hash: assign each component to grid cells
4:      for each component $i$ do
5:          $\mathbf{f}_i \leftarrow \mathbf{0}$
6:          for each $j$ in neighboring cells of $i$ do
7:              if $i \neq j$ and $d_{ij}(\mathbf{x}) < 0$ then
8:                  $\mathbf{v}_{ij} \leftarrow (x_j - x_i, y_j - y_i)/\|(x_j - x_i, y_j - y_i)\|$
9:                  $\mathbf{f}_i \leftarrow \mathbf{f}_i - |d_{ij}| \cdot \mathbf{v}_{ij}$ {Repulsion}
10:             end if
11:         end for
12:         if component $i$ outside canvas then
13:             $\mathbf{f}_i \leftarrow \mathbf{f}_i - \nabla E^{(i)}_{\text{bound}}(\mathbf{x}_i)$ {Boundary gradient}
14:         end if
15:     end for
16:     $\eta \leftarrow \min(0.1, 1.0/\sqrt{\text{iteration}})$ {Adaptive step size}
17:     $\mathbf{x} \leftarrow \mathbf{x} + \eta \cdot \mathbf{f}$
18: end while
```

ensures stable convergence. The algorithm converges when legality reaches $A_u/A_s \geq 0.99$ or when a maximum of 1000 iterations is reached. In practice, convergence is typically achieved in 15–30 iterations for IBM clustered circuits and 20–50 iterations for ISPD circuits, with an average of 23 iterations across all benchmarks. The legality threshold of 0.99 is chosen to ensure near-perfect legal placements while allowing minor numerical tolerances in floating-point arithmetic.

5 Experiments

5.1 Experimental Setup

We evaluate our energy-guided continuous diffusion approach on two standard chip placement benchmarks: ICCAD04 (IBM circuits) and ISPD2005 [15]. Following ChipDiffusion [12], we preprocess circuits by clustering standard cells into 512 macro blocks using hMetis [7], treating each cluster as a single component.

The training uses the Adam optimizer [8] with learning rate 3×10^{-4}, $\beta_1 = 0.9$, $\beta_2 = 0.999$, and $\epsilon = 10^{-8}$. We employ PPO with clipping parameter $\epsilon_{\text{clip}} = 0.2$, performing 4 PPO epochs per batch update with minibatch size of 8. The value function loss is weighted by $c_{\text{value}} = 0.5$, and gradient clipping is applied with maximum norm of 0.5. GAE parameters are $\lambda_{\text{GAE}} = 0.95$ and $\gamma = 0.99$. Training proceeds for 1000 epochs with batch size of 32 circuits, using 20,000 synthetically generated circuits with random topologies. The reward function coefficients (Eq. 10) are $\alpha_{\text{noise}} = 0.01$ and $\alpha_{\text{ent}} = 0.001$. 1000 diffusion steps are used with cosine noise schedule $\beta_t \in [0.0001, 0.02]$, following ChipDiffusion [12]. The energy function weights are $\lambda_{\text{overlap}} = 2.0$ and $\lambda_{\text{bound}} = 1.0$. The GNN encoder uses mean aggregation, and the value function shares the same encoder architecture with a separate MLP head. All results are reported on the average performance over five runs with different seeds. All experiments were conducted on a single NVIDIA RTX A6000 GPU paired with dual Intel Xeon Gold 5218R CPUs. Both training and inference use the same hardware configuration.

Table 1: IBM benchmarks: HPWL averages (clustered 512, macro-only, mixed-size). Runtime for clustered.

Method	Clust. (10^7)	Macro (10^5)	Mixed (10^6)	Time (min)
DREAMPlace [13]	3.72	–	23.6	0.48
MaskPlace [11]	–	8.72	–	–
ChiPFormer [10]	3.15	7.33	28.9	124
EfficientPlace [3]	–	8.32	–	–
WireMask-BBO [21]	–	7.43	–	–
ChipDiffusion [12]	2.98	2.49	22.7	4.4
Ours	**2.84**	**2.35**	**21.8**	**4.4**

5.2 Results

Table 1 presents our results on IBM benchmark suite, including clustered (512-cluster preprocessing via hMetis), macro-only, and mixed-size placements. Our method achieves consistent improvements across all configurations: 2.84×10^7 on clustered benchmarks (4.6% better than ChipDiffusion), 2.35×10^5 on macro-only (5.6% better), and 21.8×10^6 on mixed-size (4.0% better). For mixed-size placement, we follow ChipDiffusion's approach [12]: using our diffusion model for macro and cluster placement, then employing DREAMPlace [13] for standard cell placement with fixed macro positions. Remarkably, our zero-shot inference on clustered benchmarks completes in 4.4 minutes on average, significantly faster than ChiPFormer (124 minutes) which requires online RL fine-tuning per circuit.

Table 2: Detailed ISPD2005 results. HPWL ($\times 10^5$) and runtime (min). Results of the baselines are from ChipDiffusion [12].

Circuit	HPWL ($\times 10^5$)					Runtime (min)			
	MaskP.	ChiPF.	WireM.	ChipD.	Ours	MaskP.	ChiPF.	ChipD.	Ours
adaptec1	8.57	6.75	5.81	9.19	6.1	139	223	4.78	4.8
adaptec2	77.7	63.8	54.5	31.0	51.8	195	234	4.53	4.5
adaptec3	108	73.2	59.2	54.4	55.2	224	284	4.73	4.7
adaptec4	91.9	85.8	62.7	54.5	62.1	718	467	4.94	4.9
bigblue1	3.11	3.05	2.12	2.64	2.2	274	256	4.83	4.8
bigblue2	TO	85.8	186	38.8	**4.5**	TO	5220	122	121
bigblue3	84.0	79.2	66.2	35.9	61.5	648	494	5.13	5.1
bigblue4	TO	548	798	141	80.2	TO	1210	18.9	18.8
Average	–	116	154	**45.9**	**44.2**	–	1049	21.2	**21.1**

Table 2 presents detailed results on the eight ISPD2005 benchmark circuits (adaptec1-4, bigblue1-4), enabling direct comparison with recent state-of-the-art methods. We compare against MaskPlace [11], ChiPFormer [10], WireMask-BBO [21], and ChipDiffusion [12]. Our method achieves an average HPWL of 44.2×10^5 across all eight circuits, competitive with ChipDiffusion's 45.9×10^5 (3.7% improvement) and significantly outperforming ChiPFormer (116×10^5) and WireMask-BBO (154×10^5). Notably, MaskPlace encounters timeouts on bigblue2 and bigblue4, the two largest circuits. In terms of runtime, our approach achieves comparable efficiency to ChipDiffusion (21.1 min vs 21.2 min average), while being substantially faster than ChiPFormer (1049 min average) and WireMask-BBO (408.5 min average). This demonstrates that our unsupervised learning from synthetic data achieves strong placement quality with competitive runtime efficiency. Figure 3 shows the representative placements generated by our method.

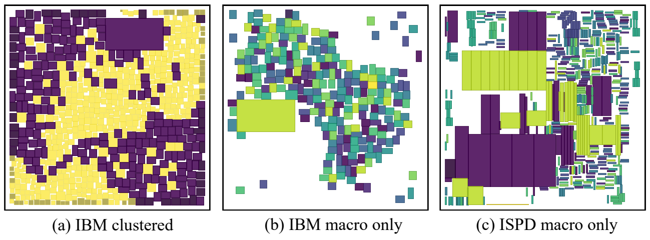

Figure 3: Representative placement results on (a) IBM clustered (512 clusters), (b) IBM macro-only, and (c) ISPD2005 benchmarks.

5.3 Ablation Studies

5.3.1 Dense vs. Sparse Energy Feedback. To validate the critical importance of computing energy at every diffusion time step, we compare our dense energy approach against a sparse-reward baseline that computes energy only at the final time step (similar to DDPO [1]). Table 3 shows that sparse energy feedback achieves only 3.42×10^7 HPWL after the same training budget, representing a 20% degradation. Moreover, sparse training exhibits 3.2× higher gradient variance and requires 2.8× more epochs to reach comparable quality. This demonstrates that dense energy feedback substantially reduces gradient variance and accelerates convergence.

Table 3: Dense vs. sparse energy feedback ablation.

Method	HPWL ($\times 10^7$)	Grad Var (relative)	Epochs to Conv.
Sparse	3.42	1.0×	2800
Dense	**2.84**	**0.31×**	**1000**

5.3.2 Legalization Decoder Contribution. We analyze the legalization decoder's impact by measuring placement quality before and after the iterative push-apart procedure (Table 4). Raw model outputs (before legalization) already achieve competitive HPWL with legality 0.953, while legalization increases HPWL by only 1.1% to achieve near-perfect legality (0.9982). This minimal degradation confirms that the diffusion model learns meaningful placement structure. The decoder only enforces hard constraints. In contrast, ChipDiffusion's gradient-based legalization requires 20,000 optimization steps with carefully tuned hyperparameters (learning rates, penalty weights).

Table 4: Decoder impact on placement quality.

Stage	HPWL ($\times 10^7$)	Legality
Raw model output	2.81	0.953
After legalization	2.84	0.9982
HPWL degradation	1.1%	

5.3.3 Penalty Weight Tuning. The energy function (Equation 5) includes penalty weights λ_{overlap} and λ_{bound} that balance wirelength optimization against constraint satisfaction. To determine optimal values, we conduct an empirical sweep over different weight configurations (Figure 4). Too small weights ($\lambda_{\text{overlap}} = 0.5, \lambda_{\text{bound}} = 0.5$) lead to insufficient constraint enforcement, resulting in poor legality (0.892) despite competitive HPWL. Too large weights ($\lambda_{\text{overlap}} = 10.0, \lambda_{\text{bound}} = 5.0$) over-emphasize constraints at the expense of HPWL optimization, increasing wirelength by 8.2% while providing minimal legality improvement beyond the 0.9982 threshold. We select $\lambda_{\text{overlap}} = 2.0$ and $\lambda_{\text{bound}} = 1.0$ as they achieve the best trade-off between HPWL and legality.

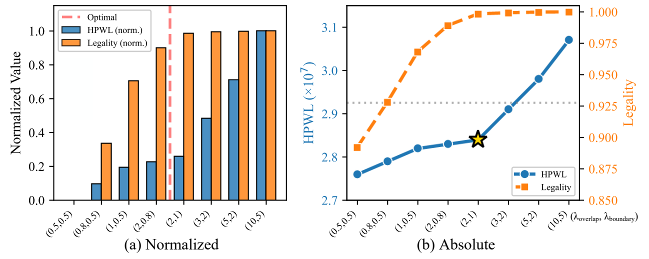

Figure 4: Penalty weights tuning. (a) Normalized HPWL and legality. (b) Absolute HPWL and legality.

5.3.4 Training Data Efficiency. We study how performance scales with synthetic training data size (Table 5). Test HPWL improves logarithmically with data: training on 5K circuits achieves 3.15×10^7 HPWL, already outperforming DREAMPlace (3.724), while 20K circuits reach our best result of 2.84×10^7. This demonstrates that our method enables effective learning even from limited synthetic data, without requiring expensive real circuit placements for supervision.

Table 5: Impact of training data size on test performance.

Training Data Size	Test HPWL ($\times 10^7$)
5K circuits	3.15
10K circuits	2.94
20K circuits	2.84
Baselines for reference:	
DREAMPlace	3.724
ChipDiffusion	2.976

6 Conclusion and Discussion

We present a policy gradient-based diffusion model framework that enables unsupervised chip placement through direct energy optimization. By extending RL-based diffusion training from discrete to continuous domains and introducing dense per-step energy feedback, our approach addresses fundamental limitations of both supervised diffusion methods and sparse-reward reinforcement learning. Although our method is slower in inference compared to analytical placers (4.4 vs. 0.48 minutes) and the scalability to thousands of macros is unvalidated, this work still demonstrates that energy-guided policy gradient diffusion can learn effective optimization strategies purely from objective functions. This opens avenues for future work in multi-objective optimization, hierarchical decomposition for large-scale circuits, and applications to other physical design tasks such as floorplanning and global routing.

References

[1] Kevin Black, Michael Janner, Yilun Du, Ilya Kostrikov, and Sergey Levine. 2024. Training Diffusion Models with Reinforcement Learning. *arXiv preprint arXiv:2305.13301* (2024).

[2] Prafulla Dhariwal and Alexander Nichol. 2021. Diffusion Models Beat GANs on Image Synthesis. In *Advances in Neural Information Processing Systems*, Vol. 34. Curran Associates, Inc., 8780–8794.

[3] Zongyue Geng, Jiahe Wang, Zhixiong Liu, Siyuan Xu, Zhengyi Tang, Min Yuan, Jing HAO, Yanzhi Zhang, and Feng Wu. 2024. Reinforcement Learning within Tree Search for Fast Macro Placement. In *Proceedings of the 41st International Conference on Machine Learning (ICML '24)*. PMLR.

[4] Jonathan Ho, Ajay Jain, and Pieter Abbeel. 2020. Denoising Diffusion Probabilistic Models. In *Advances in Neural Information Processing Systems*, Vol. 33. Curran Associates, Inc., 6840–6851.

[5] Jonathan Ho, Tim Salimans, Alexey Gritsenko, William Chan, Mohammad Norouzi, and David J. Fleet. 2022. Video Diffusion Models. *arXiv preprint arXiv:2204.03458* (2022).

[6] Andrew B. Kahng. 2024. Reevaluating Google's Reinforcement Learning for IC Macro Placement. *Commun. ACM* 67, 11 (2024), 60–71. doi:10.1145/3676845

[7] George Karypis, Rajat Aggarwal, Vipin Kumar, and Shashi Shekhar. 1999. Multilevel Hypergraph Partitioning: Applications in VLSI Domain. *IEEE Transactions on Very Large Scale Integration (VLSI) Systems* 7, 1 (1999), 69–79. doi:10.1109/92.748202

[8] Diederik P. Kingma and Jimmy Ba. 2015. Adam: A Method for Stochastic Optimization. In *International Conference on Learning Representations*.

[9] Zhifeng Kong, Wei Ping, Jiaji Huang, Kexin Zhao, and Bryan Catanzaro. 2021. DiffWave: A Versatile Diffusion Model for Audio Synthesis. In *International Conference on Learning Representations (ICLR '21)*.

[10] Yao Lai, Jinxin Liu, Zhentao Tang, Bin Wang, Jianye Hao, and Ping Luo. 2023. ChiPFormer: Transferable Chip Placement via Offline Decision Transformer. In *Proceedings of the 40th International Conference on Machine Learning (ICML '23, Vol. 202)*. PMLR, 18346–18364.

[11] Yao Lai, Yongqi Mu, and Ping Luo. 2022. MaskPlace: Fast Chip Placement via Reinforced Visual Representation Learning. In *Advances in Neural Information Processing Systems*, Vol. 35. Curran Associates, Inc., 24019–24030.

[12] Vint Lee, Minh Nguyen, Leena Elzeiny, Chun Deng, Pieter Abbeel, and John Wawrzynek. 2025. Chip Placement with Diffusion Models. In *Proceedings of the 42nd International Conference on Machine Learning (ICML '25)*. PMLR.

[13] Yibo Lin, Shounak Dhar, Wuxi Li, Haoxing Ren, Brucek Khailany, and David Z. Pan. 2019. DREAMPlace: Deep Learning Toolkit-Enabled GPU Acceleration for Modern VLSI Placement. In *Proceedings of the 56th Annual Design Automation Conference (DAC '19)*. ACM, New York, NY, USA, 1–6. doi:10.1145/3316781.3317803

[14] Azalia Mirhoseini, Anna Goldie, Mustafa Yazgan, Joe Wenjie Jiang, Ebrahim Songhori, Shen Wang, Young-Joon Lee, Eric Johnson, Omkar Pathak, Azade Nazi, Jiwoo Pak, Andy Tong, Kavya Srinivasa, William Hang, Emre Tuncer, Quoc V. Le, James Laudon, Richard Ho, Roger Carpenter, and Jeff Dean. 2021. A Graph Placement Methodology for Fast Chip Design. *Nature* 594, 7862 (2021), 207–212. doi:10.1038/s41586-021-03544-w

[15] Gi-Joon Nam, Charles J. Alpert, Paul Villarrubia, Bruce Winter, and Mehmet Can Yildiz. 2005. The ISPD2005 Placement Contest and Benchmark Suite. In *Proceedings of the 2005 International Symposium on Physical Design (ISPD '05)*. ACM, New York, NY, USA, 216–220. doi:10.1145/1055137.1055182

[16] Alexander Quinn Nichol and Prafulla Dhariwal. 2021. Improved Denoising Diffusion Probabilistic Models. In *Proceedings of the 38th International Conference on Machine Learning (ICML '21, Vol. 139)*. PMLR, 8162–8171.

[17] Sebastian Sanokowski, Wilhelm Berghammer, Martin Ennemoser, Haoyu Peter Wang, Sepp Hochreiter, and Sebastian Lehner. 2025. Scalable Discrete Diffusion Samplers. In *The Thirteenth International Conference on Learning Representations (ICLR '25)*.

[18] Sebastian Sanokowski, Sepp Hochreiter, and Sebastian Lehner. 2024. A Diffusion Model Framework for Unsupervised Neural Combinatorial Optimization. In *Proceedings of the 41st International Conference on Machine Learning (ICML '24, Vol. 235)*. PMLR, 43346–43367.

[19] John Schulman, Philipp Moritz, Sergey Levine, Michael Jordan, and Pieter Abbeel. 2015. High-Dimensional Continuous Control Using Generalized Advantage Estimation. *arXiv preprint arXiv:1506.02438* (2015).

[20] John Schulman, Filip Wolski, Prafulla Dhariwal, Alec Radford, and Oleg Klimov. 2017. Proximal Policy Optimization Algorithms. *arXiv preprint arXiv:1707.06347* (2017).

[21] Yunqi Shi, Ke Xue, Song Lei, and Chao Qian. 2023. Macro Placement by Wire-Mask-Guided Black-Box Optimization. In *Advances in Neural Information Processing Systems*, Vol. 36. New Orleans, LA, 6825–6843.

[22] Yang Song, Jascha Sohl-Dickstein, Diederik P. Kingma, Abhishek Kumar, Stefano Ermon, and Ben Poole. 2021. Score-Based Generative Modeling through Stochastic Differential Equations. In *International Conference on Learning Representations (ICLR '21)*.

[23] Richard S. Sutton and Andrew G. Barto. 2018. *Reinforcement Learning: An Introduction* (second ed.). The MIT Press, Cambridge, Massachusetts.

[24] Minkai Xu, Lantao Yu, Yang Song, Chence Shi, Stefano Ermon, and Jian Tang. 2022. GeoDiff: A Geometric Diffusion Model for Molecular Conformation Generation. In *International Conference on Learning Representations (ICLR '22)*.

[25] Ke Xue, Ruo-Tong Chen, Rong-Xi Tan, Xi Lin, Yunqi Shi, Siyuan Xu, Mingxuan Yuan, and Chao Qian. 2024. BBOPlace-Bench: Benchmarking Black-Box Optimization for Chip Placement. *arXiv preprint arXiv:2510.23472* (2024).

[26] Ke Xue, Xi Lin, Yunqi Shi, Shixiong Kai, Siyuan Xu, and Chao Qian. 2024. Escaping Local Optima in Global Placement. *arXiv preprint arXiv:2402.18311* (2024).

[27] Summer Yue, Ebrahim M. Songhori, Joe Wenjie Jiang, Toby Boyd, Anna Goldie, Azalia Mirhoseini, and Sergio Guadarrama. 2022. Scalability and Generalization of Circuit Training for Chip Floorplanning. In *Proceedings of the 2022 International Symposium on Physical Design (ISPD '22)*. ACM, New York, NY, USA, 65–70. doi:10.1145/3505170.3511478