

Article

CoDPlacer: A Fast Approach for Digital IC Placement via Consistency Distillation

Mohammad Alijanpour Shalmani *  and Jiann Shiun Yuan * 

College of Engineering and Computer Science, University of Central Florida, Orlando, FL 32816, USA

* Correspondence: alijanpour@ucf.edu (M.A.S.); jiann-shiun.yuan@ucf.edu (J.S.Y.)

Abstract

Modern digital IC placement requires methods that balance placement quality, legality, and runtime as circuit sizes continue to grow. Learning-based generative placers provide a promising alternative to analytical and sequential approaches, but they remain computationally expensive because high-quality generation requires large number of steps. This paper presents CoDPlacer, a consistency-distilled framework for accelerating generative chip placement. The method distills a pretrained diffusion placer into a consistency model for faster macro- and cluster-level placement. Two full-chip placement flows are evaluated: macro placement followed by DREAMPlace standard-cell refinement (conventional method), and a hierarchical flow using standard-cell clustering, boundary-aware expansion, local DREAMPlace optimization, and final assembly. Experiments on 18 IBM benchmarks and five ISPD2005 circuits evaluate inference-step sensitivity, macro placement, and downstream detailed placement. The 128-step consistency model achieves 4.1×, 3.11×, and 7.82× generation speedups relative to the strongest evaluated baseline on clustered IBM, macro-only IBM, and macro-only ISPD2005 benchmarks, respectively, while maintaining competitive HPWL. In conventional full-chip placement, the consistency-distilled model achieves the lowest average HPWL among the evaluated baselines. These results show that consistency distillation provides an effective quality-runtime tradeoff and useful initialization for downstream analytical placement.

Keywords: chip placement; consistency distillation; diffusion models; DREAMPlace; hierarchical placement; HPWL; hypergraph partitioning; macro placement

1. Introduction

Modern VLSI placement profoundly influences downstream routing, timing closure, and overall power–performance–area (PPA). Growing design scales, heterogeneous macros, tight physical/legalization constraints, and multi-physics objectives (e.g., thermal) make placement both computationally demanding and central to design quality. Research spans classical analytical solvers, GPU-accelerated implementations, routability-centric extensions, reinforcement learning (RL) and graph-based learning and, more recently, generative models. This section reviews representative works across these axes to motivate the methodology that follows.

Analytical and GPU-accelerated backbones. Classical analytical placers such as *RePlace* and *ePlace* use electrostatics-based density modeling and nonlinear optimization to achieve high-quality placement [1,2]. *DREAMPlace* accelerated this paradigm using GPU-based deep-learning infrastructure while maintaining competitive solution quality [3]. Subsequent works further improved efficiency through dataflow-driven GPU optimization



Received: 15 August 2026

Revised: 23 September 2026

Accepted: 30 September 2026

Published: 3 October 2026

Copyright: © 2026 by the authors.

Licensee MDPI, Basel, Switzerland.

This article is an open access article distributed under the terms and

conditions of the [Creative Commons Attribution \(CC BY\)](https://creativecommons.org/licenses/by/4.0/) license.

in *DG-RePlace* [4], concurrent detailed placement in *ABCDPlace* [5], and operator fusion and optimized scheduling in *XPlace* [6]. More recent mixed-size placement methods also incorporate second-order information to improve optimization stability and the runtime–quality trade-off [7].

Routability and congestion awareness. Recent placement methods increasingly incorporate routability into optimization. *DrPlace* combines routability-aware objectives with legalization and detailed placement to improve post-route quality [8], while *GOALPlace* learns density targets from post-route feedback to reduce DRC and timing degradation [9]. Other approaches model pin-access constraints directly during placement using Boolean satisfiability to improve routability [10].

RL and sequential decision-making. RL- and CNN-based methods have shown strong performance across domains such as autonomous driving, control, action recognition, and object detection [11–14], motivating their use in chip placement and routing. Early approaches such as *DeepPlace* and *DeepPR* applied RL and learned representations to placement and routing [15], while subsequent studies examined the generalization and reproducibility of macro-placement RL [16,17]. Other works improved efficiency and stability through tree-search-guided RL and macro-regulation strategies [18,19], or combined RL with conventional placement and routing engines in *AutoDMP* and *RoutePlacer* [20,21]. More recent methods include visual RL in *MaskPlace* [22], transferable offline decision learning in *ChiPFormer* [23], and policy-gradient placement coupled with generative routing [24].

Graph-based learning and transferable guidance. GNNs have been successfully applied across domains including recommender systems and protein modeling [25,26], motivating their use in placement. *PL-GNN* and its affinity-aware extension use graph structure to guide cell clustering and placement, improving HPWL and other design metrics [27,28]. *TransPlace* targets cross-design generalization through transferable embeddings [29], while *Graph-of-Circuits* uses GNNs for physical-design exploration [30]. Other approaches incorporate PPA objectives into graph clustering [31] or use lightweight graph-signal processing, as in *GiFt*, to accelerate placement without heavy GNN training [32].

Generative and emerging paradigms. *DREAM-GAN* injects adversarial learning into *DREAMPlace* to generate density /placement patterns aligned with high-quality solutions [33]. More recently, *ChipDiffusion* formulated chip placement as a diffusion-based generative process, where a denoising network learns to generate placement solutions from noisy spatial representations while incorporating circuit connectivity and placement priors [34]. This formulation is attractive because it avoids purely sequential placement decisions and can generate placements for many objects jointly. However, although diffusion-based placement can offer runtime advantages over sequential learning-based approaches, its inference remains time-consuming and computationally expensive: high-quality sampling typically requires many reverse denoising steps, and the model used in this work follows a 1000-step denoising trajectory. Such latency is a major limitation for placement, where inference may need to be repeated across circuits, seeds, samples, legalization settings, and design-space exploration loops. Table 1 summarizes representative related works in placement research, including their main methodological categories, contributions, and evaluation benchmarks.

To address this bottleneck, recent *consistency models* introduce a faster generative paradigm in which a model learns to map noisy samples at different noise levels directly to a consistent clean solution [35]. In particular, *consistency distillation* transfers the behavior of a pretrained diffusion teacher into a student model that supports one-step or few-step generation, while retaining much of the teacher model’s learned distribution. Motivated by the importance of runtime in electronic design automation, this paper includes three main

contributions. First, we introduce a consistency-distilled generative placement model that reduces the inference cost of placement while preserving competitive HPWL and legality. Second, we systematically evaluate the effect of consistency inference steps on placement quality and runtime across benchmarks. Third, we integrate the distilled model into both a conventional macro-placement flow and a hierarchical full-chip placement flow using standard-cell clustering, boundary-aware expansion, and DREAMPlace refinement. All codes and data preparation scripts are available at <https://github.com/mohammad-AJP/CoDPlacer>, accessed on 12 August 2026.

Table 1. Summary of related works in placement research.

Paper	Category	Main Contribution	Benchmarks/Context
RePlace [1]	Analytic	Nesterov + electrostatics density	ISPD'05/'06
ePlace [2]	Analytic	Electrostatics density + step control	ISPD'05/'06
DREAMPlace [3]	GPU/Analytic	PyTorch GPU; $\sim 30\times$ speedup	ISPD'05/'06
ABCDPlace [5]	GPU/Detailed	Batch concurrent detailed placement	ISPD
DG-RePlace [4]	GPU/Analytic	Dataflow-driven GPU acceleration	ISPD
XPlace [6]	GPU/Analytic	Operator fusion; FFT density	ISPD
Stronger Backbone [7]	Analytic	Second-order mixed-size backbone	ISPD
DrPlace [8]	Routability	Pin-density; routability-aware DP	ISPD/DAC
GOALPlace [9]	Routability+Learning	Empirical-Bayes density targets	Post-route guided
Routing-Aware SAT [10]	Routability	Pin-access constraints in placement	Custom SAT
DeepPlace/DeepPR [15]	RL/Joint	End-to-end place+route via RL	Public benchmarks
RL assessments [16,17]	RL	Generalization/reproducibility studies	TPU/public
RL Tree Search [18]	RL	Efficient guided exploration	Public
RL Macro Regulator [19]	RL	RL as controller for macros	Benchmarks
AutoDMP [20]	RL + DREAMPlace	Automated macro pipeline	Public/industrial
RoutePlacer [21]	RL + Routing	Router-in-the-loop rewards	Academic
MaskPlace [22]	RL (Visual)	Pixel-level RL; 0% overlap	NeurIPS'22
ChiPFormer [23]	Offline DT	Transferable macro placement	Macro datasets
PolicyGrad+GenRoute [24]	RL/Joint	Policy gradient + generative routing	NeurIPS'22
PL-GNN [27]	GNN	Affinity-aware clustering guidance	Industrial
ISPD'21 Affinity [28]	GNN	Affinity-aware extension	ISPD'21
TransPlace [29]	GNN	Transferable embeddings	Academic
Graph-of-Circuits [30]	GNN	GNN exploration of design space	NeurIPS'23
PPA Graph Clustering [31]	GNN	PPA-aware unsupervised clustering	Public/industrial
GiFt [32]	GSP	Parameter-free graph filters	ICCAD'24
DREAM-GAN [33]	Generative	Adversarial density/placement priors	Public
Diffusion Placement [34]	Generative	Diffusion for chip placement	Public

2. Materials and Methods

2.1. Data

This study adopts the data representation and benchmark-preparation pipeline introduced in ChipDiffusion [34]. Instead of training a diffusion model from scratch, we use the pretrained ChipDiffusion model as the teacher and apply consistency distillation using the same graph-based placement data employed in [34]. The training data are synthetically generated by sampling circuit size, object dimensions, aspect ratio, placement density, terminal configuration, and hierarchical connectivity to construct diverse legal placement instances. Each circuit is represented as a graph in which nodes correspond to placeable objects and edges represent circuit connectivity. Node features include physical object dimensions, while edge attributes encode pin-offset information. The placement target is the two-dimensional lower-left coordinate of each object.

During consistency distillation, noisy placement states are generated from the training placements at sampled diffusion timesteps. The frozen diffusion teacher performs a deterministic ODE transition between adjacent noise levels, and the student is optimized to produce consistent placement predictions across these states. Consequently, the distilled

model can generate placements using one or a small number of inference steps rather than the long iterative denoising trajectory required by the original diffusion model. The objective is to determine whether this reduction in inference cost can be achieved while retaining useful placement quality.

For evaluation, we use public physical-design benchmarks from the IBM [36] and ISPD [37] suites. These benchmarks provide object dimensions, pin locations, fixed terminals, and net connectivity. To make large circuits compatible with the graph-based generative model, we follow a clustering procedure described in Section 2.2. Standard cells are grouped through hypergraph partitioning, whereas macros and fixed terminals remain individual nodes. The resulting reduced circuit graph therefore contains standard-cell clusters, macros, and ports or fixed terminals.

2.2. Standard-Cell Clustering

To reduce the size of the placement problem and enable hierarchical full-chip placement, we grouped the standard cells of each circuit using two different clustering models: KaHyPar (Karlsruhe Hypergraph Partitioning), a multilevel hypergraph partitioning framework that divides a hypergraph into balanced clusters while minimizing connectivity between the resulting partitions [38], and hMETIS, a multilevel hypergraph partitioning method developed for VLSI applications [39]. However, due to satisfactory results of hMetis and also to have a fair comparison, we decided to move forward with hMetis 1.5.3. The circuit netlist is modeled as a hypergraph, where standard cells correspond to vertices and nets correspond to hyperedges. hMETIS partitions this hypergraph into balanced groups while minimizing the number of cut hyperedges, which helps preserve local connectivity inside each cluster. To be able to compare results with [34], we used 512 standard-cell clusters for each IBM benchmark circuit. Macros are not absorbed into the clusters; they remain explicit placement objects so that the learned model can jointly place macros and coarse standard-cell regions.

After clustering, each cluster is represented as a movable object in the global placement problem. The consistency-distilled model predicts the locations of both macros and standard-cell clusters on the chip canvas. This evaluation setup separates two questions. First, the clustered IBM graphs evaluate the quality and speed of the learned consistency model at the coarse placement level. Second, expanded or refined placements can be used to evaluate whether the generated cluster-level solutions remain useful after reconstructing standard-cell-level layouts. This distinction is important because cluster-level HPWL measures the learned global placement quality, whereas full expanded placement quality reflects practical downstream usability.

2.3. Proposed Method

The proposed method applies consistency distillation to graph-based chip placement. Let a circuit be represented as a graph

$$\mathcal{G} = (\mathcal{V}, \mathcal{E}), \quad (1)$$

where each node $i \in \mathcal{V}$ denotes a placeable object, such as a macro, a standard-cell cluster, or a terminal, and each edge represents circuit connectivity. The placement is represented by

$$\mathbf{x}_0 \in \mathbb{R}^{|\mathcal{V}| \times 2}, \quad (2)$$

where each row gives the normalized two-dimensional center coordinate of one object. Lower-left benchmark coordinates are converted to object centers during preprocessing

and converted back during postprocessing. Normalized object sizes and center-relative pin offsets are provided as graph-conditioning attributes.

We used the model in [34] as a teacher and distill it into a consistency model. The goal is to teach a student model that maps a noisy placement directly to a clean placement estimate in one or a few inference steps. This is different from conventional iterative denoising, where many reverse steps are required.

2.3.1. Noising Process

Given a clean placement $\mathbf{x}_0$, a noisy placement at continuous time $t \in [0, 1]$ is constructed as

$$\mathbf{x}_t = \alpha(t)\mathbf{x}_0 + \sigma(t)\epsilon, \quad \epsilon \sim \mathcal{N}(\mathbf{0}, \mathbf{I}), \quad (3)$$

where the cosine schedule is used:

$$\alpha(t) = \cos\left(\frac{\pi t}{2}\right), \quad \sigma(t) = \sin\left(\frac{\pi t}{2}\right). \quad (4)$$

Large values of t correspond to highly noisy placements, while small values of t correspond to nearly clean placements.

2.3.2. Consistency Function

The student model learns a consistency function

$$f_\theta(\mathbf{x}_t, t, \mathcal{G}) \approx \mathbf{x}_0, \quad (5)$$

which predicts the clean placement from any noisy point along the same denoising trajectory. To enforce the boundary condition

$$f_\theta(\mathbf{x}_0, 0, \mathcal{G}) = \mathbf{x}_0, \quad (6)$$

we use the following parameterization:

$$f_\theta(\mathbf{x}_t, t, \mathcal{G}) = c_{\text{skip}}(t)\mathbf{x}_t + c_{\text{out}}(t)F_\theta(\mathbf{x}_t, t, \mathcal{G}), \quad (7)$$

where F_θ is the graph neural network backbone. The scaling terms are defined as

$$c_{\text{skip}}(t) = \frac{\alpha(t)^2\sigma_{\text{data}}^2}{\sigma(t)^2 + \alpha(t)^2\sigma_{\text{data}}^2}, \quad (8)$$

and

$$c_{\text{out}}(t) = \frac{\sigma(t)\sigma_{\text{data}}}{\sqrt{\sigma(t)^2 + \alpha(t)^2\sigma_{\text{data}}^2}}. \quad (9)$$

In our implementation, $\sigma_{\text{data}} = 0.5$. This parameterization gives $c_{\text{skip}}(0) = 1$ and $c_{\text{out}}(0) = 0$, so the model naturally satisfies the clean-placement boundary condition.

2.3.3. Consistency Distillation

During training, we sample adjacent timesteps t_n and t_{n+1} from a discretized schedule, where $t_n > t_{n+1}$. First, a noisy placement $\mathbf{x}_{t_n}$ is generated from the ground-truth placement $\mathbf{x}_0$. The frozen teacher model predicts the noise component:

$$\hat{\epsilon}_\phi = T_\phi(\mathbf{x}_{t_n}, t_n, \mathcal{G}), \quad (10)$$

where T_ϕ denotes the teacher model. A Denoising Diffusion Implicit Model (DDIM) style step is then used to estimate the corresponding lower-noise sample:

$$\hat{\mathbf{x}}_0 = \frac{\mathbf{x}_{t_n} - \sigma(t_n)\hat{\epsilon}_\phi}{\alpha(t_n)}, \quad (11)$$

$$\hat{\mathbf{x}}_{t_{n+1}} = \alpha(t_{n+1})\hat{\mathbf{x}}_0 + \sigma(t_{n+1})\hat{\epsilon}_\phi. \quad (12)$$

The student prediction at the higher-noise point is

$$\mathbf{y}_\theta = f_\theta(\mathbf{x}_{t_n}, t_n, \mathcal{G}), \quad (13)$$

and the target is produced by an exponential moving average (EMA) version of the student:

$$\mathbf{y}_{\theta^-} = f_{\theta^-}(\hat{\mathbf{x}}_{t_{n+1}}, t_{n+1}, \mathcal{G}). \quad (14)$$

The consistency objective minimizes the coordinate-wise pseudo-Huber distance between the student prediction and the stop-gradient EMA target:

$$\mathcal{L}_{\text{CD}} = \mathbb{E}_{\mathbf{x}_0, n, \epsilon} \left[\frac{1}{2|\mathcal{M}|} \sum_{i \in \mathcal{M}} \sum_{d \in \{x, y\}} \left(\sqrt{(y_{\theta, i, d} - y_{\theta^-, i, d})^2 + c^2} - c \right) \right], \quad (15)$$

where $\mathcal{M}$ is the set of movable objects, d indexes the horizontal and vertical placement coordinates, and $c = 10^{-3}$ is the pseudo-Huber constant. The EMA prediction is treated as a stop-gradient training target. Objects marked as fixed ports or terminals by the placement mask are excluded from the loss and retain their original positions. Applying the pseudo-Huber penalty independently to each coordinate reduces sensitivity to large prediction errors compared with a coordinate-wise mean-squared-error objective.

The exponential moving average (EMA) parameters are updated after each optimization step:

$$\theta^- \leftarrow \mu\theta^- + (1 - \mu)\theta, \quad (16)$$

where μ is linearly increased during training from 0.95 to 0.999.

2.3.4. Training Configuration

The consistency model uses the same graph neural network backbone as the teacher model, but only the student backbone is optimized. The teacher network is frozen throughout training. The student and EMA networks are initialized from the teacher backbone weights before distillation. We train the consistency model using Adam with learning rate 10^{-4} , batch size 64, and 1,000,000 training steps. For consistency distillation, the number of training timestep pairs is fixed to $N = 18$.

2.3.5. Inference

At inference time, the model starts from Gaussian noise:

$$\mathbf{x}_{t_1} \sim \mathcal{N}(\mathbf{0}, \mathbf{I}). \quad (17)$$

For one-step generation, the placement is obtained directly as

$$\hat{\mathbf{x}}_0 = f_\theta(\mathbf{x}_{t_1}, t_1, \mathcal{G}). \quad (18)$$

For multi-step generation, the model alternates between predicting a clean placement and re-noising it to the next timestep:

$$\hat{\mathbf{x}}_0^{(k)} = f_{\theta}(\mathbf{x}_{t_k}, t_k, \mathcal{G}), \quad (19)$$

$$\mathbf{x}_{t_{k+1}} = \alpha(t_{k+1})\hat{\mathbf{x}}_0^{(k)} + \sigma(t_{k+1})\epsilon, \quad \epsilon \sim \mathcal{N}(\mathbf{0}, \mathbf{I}). \quad (20)$$

The final clean prediction is used as the generated placement. In this work, we evaluate various steps of consistency sampling. Fixed ports and terminals are preserved during sampling by applying the same placement mask used during training.

The main advantage of this formulation is that the number of neural network evaluations is reduced from 1000 iterative denoising steps to a small fixed number of consistency evaluations. This allows us to study the tradeoff between placement quality and inference cost while keeping the benchmark representation and teacher model fixed.

2.3.6. Detailed Placement Strategies

To evaluate the proposed consistency-distilled placement model beyond the macro-only setting, we consider two full-placement strategies. The first strategy follows the conventional evaluation protocol used in many macro-placement works. In this flow, the learned model is used only to generate macro locations. These macros are then inserted into the original full-chip file and fixed at the predicted locations, while all standard cells remain movable. A conventional analytical placer, DREAMPlace (<https://github.com/limbo018/DREAMPlace> accessed on 12 August 2026), is then used to place, legalize, and refine the standard cells around the fixed macros. This setting evaluates whether the learned macro placement provides a useful initialization for a standard downstream placement engine. The flow of this approach is shown in Figure 1.

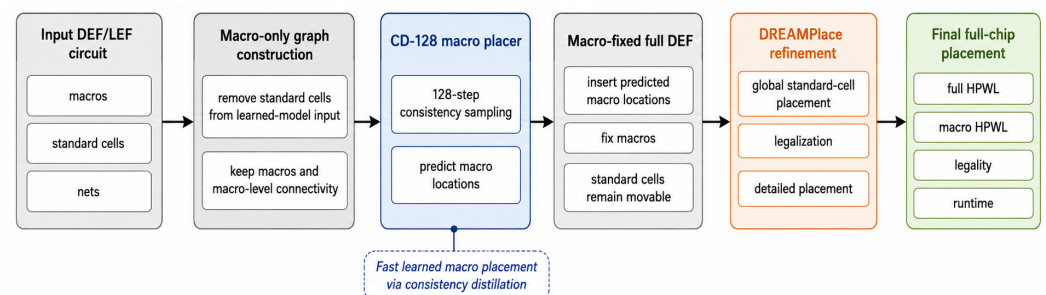


Figure 1. Architecture of the CD-128 + DREAMPlace full-placement flow. The consistency-distilled model is used to predict macro locations from a macro-only graph, after which the predicted macros are fixed in the original full-chip. DREAMPlace then performs global standard-cell placement, legalization, and detailed placement around the fixed learned macros which follows the common macro-placement evaluation flow.

To extend the learned macro/cluster placement toward full-chip placement, we also use a hierarchical flow composed of three stages as depicted in Figure 2. First, the consistency-distilled model predicts the global locations of macros and clusters on the original chip canvas. Each cluster represents a group of standard cells and is treated as a soft placement object during this global stage. Second, each cluster is expanded into a local DREAMPlace problem containing the standard cells assigned to that cluster. DREAMPlace is then used to optimize the standard-cell locations inside each cluster independently. Third, the locally optimized cluster placements are mapped back to the original full-chip coordinate system using the predicted global cluster locations, and a final full-chip placement file is generated for legalization and evaluation.

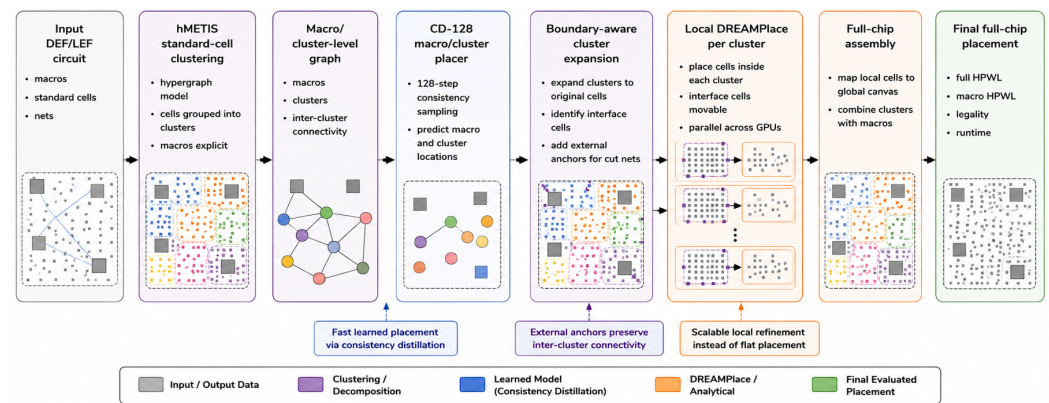


Figure 2. Architecture of the hierarchical CD-128 full-placement flow. Standard cells are first grouped into clusters (different cluster colors are used only to distinguish clusters and carry no quantitative meaning) using hMETIS, while macros remain explicit placement objects. The consistency-distilled model predicts global macro and cluster locations, after which each cluster is expanded into its original standard cells and optimized locally using DREAMPlace with boundary-aware interface-cell handling. The locally optimized clusters are then mapped back to the full-chip coordinate system and assembled into a complete placement for evaluation.

Figures 1 and 2 represent two alternative full-placement flows that share the same proposed consistency-distilled global placement model. In Figure 1, CD-128 is used only for macro placement, followed by conventional DREAMPlace standard-cell placement, legalization, and detailed placement. In Figure 2, CD-128 jointly places macros and standard-cell clusters within a hierarchical flow, followed by boundary-aware cluster expansion and local DREAMPlace refinement. Thus, the main optimization introduced in this work is the learned global-placement stage, where the consistency distillation is used as placer, while the downstream DREAMPlace stages remain conventional.

2.4. Boundary-Aware Cluster Expansion

A key issue in such a hierarchical flow is that clusters are not electrically independent. Nets crossing cluster boundaries can be poorly represented if each cluster is optimized in isolation. To reduce this loss of global connectivity information, we use the boundary-aware cluster expansion. For every cut net leaving a cluster, a fixed external anchor is inserted into the local DREAMPlace instance to represent the estimated position of the external endpoint. Standard cells connected to such cut nets are marked as interface cells. These interface cells are not fixed; instead, they are initialized near the cluster boundary indicated by their external anchors and remain movable during local DREAMPlace optimization. This provides a soft boundary constraint: the local placer sees the direction of external connections while retaining enough freedom to optimize local wirelength and legality.

As circuits get bigger in size, the computational cost of their placement skyrockets. The main advantage of this hierarchy is scalability. Instead of solving the entire full-chip standard-cell placement problem in one flat optimization, the design is decomposed into many smaller cluster-local placement problems. These local problems can be solved independently and in parallel across multiple GPUs or CPU workers. As a result, the approach reduces the memory and compute burden of standard-cell refinement and creates a practical path for applying learned global placement models to larger circuits. At the same time, the final assembly and legalization stages preserve compatibility with conventional placement evaluation.

3. Results

This section reports the placement results on IBM and ISPD benchmark dataset. The evaluation includes the diffusion teacher and consistency-distilled models with different inference budgets. Lower HPWL values indicate better wirelength. Legality and macro legality are reported as normalized scores. Legality is measured as the ratio between the non-overlapping object area inside the chip boundary and the total object area. A score of 1.0 indicates that all evaluated objects are within the placement region and have no overlaps, while lower values indicate overlap or boundary violations. Experiments were conducted on an Ubuntu 22.04.5 machine with an Intel Core i9 CPU, 128 GB of RAM and an NVIDIA GeForce RTX 2080 Ti GPU with 11 GB memory. All consistency distillation and model evaluation used a single GPU, with Python 3.10, PyTorch 2.2.1, CUDA 12.1, and PyTorch Geometric 2.4.0; downstream placement was performed using DREAMPlace.

3.1. Benchmark Statistics

Tables 2 and 3 summarize the principal characteristics of the IBM and ISPD2005 benchmarks used in this study, including the numbers of objects, movable or standard cells, macros and fixed terminals, nets, and pins. Together, these benchmarks cover circuits ranging from relatively small designs to instances containing more than two million objects, enabling evaluation across different placement scales and connectivity complexities. We only used the first five circuits of ISPD2005 benchmark and reported the results on only these circuits.

Table 2. Specifications of the IBM placement benchmarks used in this study.

Benchmark	Total Objects	Standard Cells	Macros/Fixed Terminals	Nets	Pins
ibm01	12,752	12,506	246	14,111	50,566
ibm02	19,601	19,330	271	19,584	81,199
ibm03	23,136	22,846	290	27,401	93,573
ibm04	27,507	27,212	295	31,970	105,859
ibm05	29,347	29,347	0	28,446	126,308
ibm06	32,498	32,320	178	34,826	128,182
ibm07	45,926	45,635	291	48,117	175,639
ibm08	51,309	51,008	301	50,513	204,890
ibm09	53,395	53,142	253	60,902	222,088
ibm10	69,429	68,643	786	75,196	297,567
ibm11	70,558	70,185	373	81,454	280,786
ibm12	71,076	70,425	651	77,240	317,760
ibm13	84,199	83,775	424	99,666	357,075
ibm14	147,605	146,991	614	152,772	546,816
ibm15	161,570	161,177	393	186,608	715,823
ibm16	183,484	183,026	458	190,048	778,823
ibm17	185,495	184,735	760	189,581	860,036
ibm18	210,613	210,328	285	201,920	819,697

Table 3. Specifications of the ISPD2005 placement benchmark.

Benchmark	Total Objects	Movable Cells	Macros/Fixed Terminals	Nets	Pins
adaptec1	211,447	210,904	543	221,142	944,053
adaptec2	255,023	254,457	566	266,009	1,069,482
adaptec3	451,650	450,927	723	466,758	1,875,039
adaptec4	496,045	494,716	1329	515,951	1,912,420
bigblue1	278,164	277,604	560	284,479	1,144,691
bigblue2	557,866	534,782	23,084	577,235	2,122,282
bigblue3	1,096,812	1,095,514	1298	1,123,170	3,833,218
bigblue4	2,177,353	2,169,183	8170	2,229,886	8,900,078

Each IBM circuit was partitioned into 512 standard-cell clusters, while macros and fixed terminals were retained as individual graph vertices. Consequently, the reduced graphs contain between 512 and 1298 vertices, substantially fewer than the original circuits containing up to 210,613 objects.

Table 4 compares the performance of the diffusion model and [34] with consistency-distilled samplers on clustered IBM dataset (each circuit is divided into macros + 512 standard cell clusters). The diffusion teacher obtains the average HPWL of 28.15 on all 18 circuits (HPWL ratio of 1.058 relative to the clustered reference placement derived from the original IBM coordinates.). However, this quality requires 170.42 s of total generation time. To have a fair comparison, the results are reported by averaging performance using five random seeds. The performance of long-step consistency sampler and their runtimes are relatively stable across all seeds.

Table 4. Performance comparison on clustered IBM benchmarks between the diffusion model and consistency-distilled models. HPWL is reported in 10^6 . Lower is better for HPWL, HPWL ratio, HPWL ratio std, and inference time; higher is better for legality and inference speedup.

Method	Steps	HPWL	HPWL Ratio	Ratio Std	Legality	Macro Leg.	Inf. Time	Inf. Speedup
Diffusion teacher	1000	28.159	1.0583	0.0553	0.997186	0.997525	170.42	1.0
Consistency model	1	82.488	2.9753	0.3334	0.967564	0.978414	27.80	6.1
Consistency model	2	38.315	1.4287	0.0946	0.996786	0.997238	26.30	6.5
Consistency model	4	34.289	1.2828	0.0764	0.999491	0.999557	26.62	6.4
Consistency model	8	33.905	1.2668	0.0936	0.999056	0.999150	26.84	6.3
Consistency model	16	33.220	1.2350	0.0785	0.998491	0.998450	26.80	6.4
Consistency model	64	31.042	1.1550	0.0726	0.998051	0.999020	33.45	5.1
Consistency model	128	28.973	1.0902	0.0619	0.998497	0.998931	41.10	4.1

The consistency-distilled model substantially reduces inference cost. The one-step model is a very fast configuration, with 27.80 s inference time and a $6.1 \times$ inference speedup, but this model achieves an average HPWL of 82.488 (HPWL ratio degrades to 2.975). This indicates that one-step sampling is too aggressive for high-quality placement in the current model and also reflects a limitation: a single evaluation must map an almost pure-noise placement directly to a globally connected and legal layout, whereas multistep sampling provides repeated lower-noise correction and guidance. Increasing the number of consistency steps sharply improves HPWL: by only changing steps to 2, the model reduces the HPWL to 38.315 (HPWL ratio to 1.429) which is a huge improvement. Longer samplers continue to close the quality gap. Setting the steps to 128 leads the model to have a competitive average HPWL of 28.973 (HPWL ratio of 1.0902) while requiring almost a quarter of diffusion model inference time plus maintaining high legality. The results show a clear quality-runtime trade-off: four to eight steps provide the strongest practical speed-quality compromise, whereas 64 to 128 steps are preferable when HPWL is prioritized. We used the consistency distillation with 128 steps (CD-128) as the main model in this work and reported the results for this model.

3.2. Inference-Step Ablation

Table 4 also isolates the effect of the number of consistency inference steps. The saturation behavior of the model by moving from 1 to 128 steps suggests that the consistency model rapidly approaches a useful placement estimate after only a few evaluations, while additional steps mainly refine wirelength.

The legality metrics remain stable after two steps. In particular, the four-step and eight-step models both exceed 0.999 or remain very close to it for cluster and macro legality. This is important because the observed runtime gains do not come from producing highly

illegal placements. Instead, the main trade-off is between HPWL quality and inference budget. Per-circuit results for all 18 clustered IBM benchmarks are provided in Table A1.

Table 5 presents a matched-NFE comparison between the consistency-distilled model and the DDIM diffusion teacher, where both methods use the same number of sampling steps. Across most NFE settings, the CD model achieves lower HPWL than DDIM, with particularly large improvements at very low step counts, such as 1 and 2 steps. At 16 steps, the two methods obtain nearly identical HPWL, while CD again achieves lower HPWL at 64 and 128 steps. The model runtimes are comparable at each matched step count, whereas the total inference time of CD is generally lower than that of DDIM for all settings except the 1-step case. These results indicate that the advantage of consistency distillation is not solely due to using fewer sampling steps; even under matched computational budgets, CD generally preserves or improves placement quality while maintaining competitive inference cost.

Table 5. Matched-step comparison between the consistency-distilled model and the DDIM diffusion teacher on clustered IBM benchmarks. HPWL is reported in 10^6 , and runtime is reported in seconds. Model time measures sampling, while total inference time includes the remaining generation-pipeline overhead. Lower is better.

Steps	CD HPWL	DDIM HPWL	CD Model Time	DDIM Model Time	CD Total Inf.	DDIM Total Inf.
1	82.488	107.470	0.17	0.15	27.80	27.40
2	38.315	42.588	0.29	0.28	26.30	27.23
4	34.289	35.264	0.51	0.54	26.62	27.81
8	33.905	33.165	1.01	1.13	26.84	28.39
16	33.220	33.232	1.92	2.01	26.80	29.84
64	31.042	31.652	7.93	7.56	33.45	34.39
128	28.973	29.634	15.77	16.21	41.10	42.84

Moreover, based on the pure model sampling time in Table 5, CD-128 provides approximately a $9\times$ speedup over Diffusion-1000, demonstrating that consistency distillation substantially reduces the actual denoising/sampling cost. The smaller speedup observed in total inference time is primarily due to the fixed generation-pipeline overhead shared by both methods.

3.3. Macro-Only Placement Performance

Table 6 compares the macro placement performance with representative macro-placement baselines on the IBM benchmarks. Compared with the diffusion model, the distilled model reduces the average runtime from 1.37 min to 0.44 min, corresponding to approximately $3.1\times$ speedup, while maintaining full macro legality. This acceleration comes with a small HPWL increase, from 2.61 to 2.79 in units of 10^5 . The result highlights the central tradeoff of the proposed approach: consistency distillation substantially reduces inference cost, which is important for placement workflows requiring repeated sampling or design-space exploration, but the faster sampler does not fully preserve the macro-HPWL quality of the diffusion teacher.

Table 7 compares the proposed 128-step consistency-distilled model with macro-placement baselines on ISPD2005 benchmark. To ensure a fair comparison, we report results only on the first five ISPD2005 circuits for which complete evaluations were successfully obtained. The consistency-distilled model achieves an average macro HPWL of 41.08, which is higher than ChipDiffusion but lower than MaskPlace and ChiPFormer. Its main advantage is runtime: the proposed model completes macro placement in 0.56 min, compared with 4.38 min for ChipDiffusion, 246 min for ChiPFormer, and 253 min for MaskPlace. This result shows the expected quality-runtime tradeoff of consistency distillation: the

model sacrifices some HPWL quality relative to the full diffusion baseline but provides a substantially faster sampling process. Therefore, the ISPD2005 results support the central motivation of this work: using consistency distillation to make diffusion-based placement more practical when placement speed is a critical constraint.

Table 6. Macro-level comparison with macro-placement baselines on IBM benchmarks. Macro HPWL values are reported in 10^5 . Lower HPWL and runtime are better.

Method	Macro HPWL (10^5)	Runtime (min)
MaskPlace	8.34	143.00
ChiPFormer	6.93	109.00
ChipDiffusion	2.61	1.37
Consistency-distilled model, 128-step	2.79	0.44

Table 7. Macro-level comparison with macro-placement baselines on ISPD2005 benchmarks. Macro HPWL values are reported in 10^5 . Lower HPWL and runtime are better.

Method	Macro HPWL (10^5)	Runtime (min)
MaskPlace	50.41	253.00
ChiPFormer	41.63	246.00
ChipDiffusion	38.10	4.38
Consistency-distilled model, 128-step	41.08	0.56

Result are computed over the five ISPD2005 samples (Adaptec1-4, BigBlue1) that completed successfully in the current run.

3.4. Detailed Placement

Table 8 evaluates whether macro placements generated by the various versions of diffusion model, the proposed 128-step consistency-distilled model, and MaskPlace are useful for a conventional full-placement flow. On average, consistency distillation with 128 steps (CD128) + DREAMPlace achieves the lowest final HPWL of 26.98×10^6 , compared with 28.45×10^6 for the diffusion model with 1000 denoising steps and 29.91×10^6 for MaskPlace. To have a fair comparison, we also evaluated the stochastic and DDIM Diffusion model with 128 denoising steps, similar to 128 steps for consistency Distillted model. The stochastic Diffusion with 128 steps achieved an average HPWL of 28.80 over five seeds. DDIM diffusion model with 128 steps also achieved an average HPWL of 28.91. The results shows that by reducing denoising steps from 1000 to 128, the performance of the model degrades. However, CD with 128 steps shows much better performance than various diffusion models. Although CD128 has higher macro-only HPWL than the diffusion model, the full-placement result is not determined solely by macro-only HPWL. The downstream DREAMPlace stage optimizes the standard-cell placement over the complete netlist, and the CD128 macro configuration appears to provide a more favorable initialization for standard-cell spreading and legalization. We have also reported the performance of pure DREAMPlace on the IBM dataset, where it achieved an average HPWL of 27.18, which is better than the diffusion variants but slightly worse than CD-128. These results suggest that consistency distillation can provide a practical learned macro-initialization stage: it preserves enough placement structure for DREAMPlace to produce competitive final layouts, while avoiding the high sampling cost of diffusion-based macro generation.

Table 9 shows that the proposed **CD-128 + DP** flow achieves the lowest average detailed-placement HPWL across the evaluated ISPD2005 benchmarks (first five circuits), with an average of 129.62×10^6 , compared with 142.16×10^6 for Diffusion-1000 + DP and 141.86×10^6 for Diffusion-128 + DP. This corresponds to an average HPWL reduction of approximately **8.8%** and **8.6%**, respectively. In particular, CD-128 achieves substantially lower HPWL on *adaptec3* and *adaptec4*, while the diffusion-based variants perform better on

some individual benchmarks such as *adaptec1*, *adaptec2*, and *bigblue1*. The results indicate that consistency distillation can preserve competitive placement quality while improving the average full detailed-placement HPWL over the evaluated ISPD2005 designs.

Table 8. Full detailed placement comparison between MaskPlace, diffusion-teacher, 128-step diffusion, diffusion with 128 DDIM steps and 128-step consistency-distilled macro placement followed by DREAMPlace for standard cell placement. HPWL is reported in 10^6 . Lower is better.

Benchmark	MP	Diff	Diff-128	DDIM-128	Pure DP	CD
ibm01	3.28	3.06	2.92	3.06	2.34	2.97
ibm02	6.12	6.27	6.44	6.45	4.68	6.28
ibm03	9.67	10.02	10.05	10.55	6.53	8.76
ibm04	9.92	9.73	9.26	9.12	8.47	9.30
ibm05	9.07	9.07	9.07	9.07	9.07	9.07
ibm06	8.28	8.53	8.75	8.26	5.71	6.92
ibm07	12.93	13.08	13.29	12.18	10.67	12.10
ibm08	14.33	15.89	16.15	14.70	12.97	15.68
ibm09	17.04	15.95	16.19	16.41	12.01	14.83
ibm10	43.46	43.14	45.15	43.60	42.75	36.81
ibm11	24.57	23.18	23.33	24.25	22.25	21.79
ibm12	50.80	44.18	44.22	47.25	39.51	42.48
ibm13	29.57	29.55	28.80	27.79	26.78	28.55
ibm14	51.08	45.58	46.91	51.05	39.00	44.77
ibm15	60.74	54.99	57.71	57.17	48.76	53.17
ibm16	73.13	67.47	66.99	64.19	58.60	61.66
ibm17	72.46	71.34	71.77	74.34	95.42	69.28
ibm18	41.98	41.05	41.38	41.01	43.63	41.26
Average	29.91	28.45	28.80	28.91	27.18	26.98

Table 9. Full detailed placement comparison on the evaluated ISPD2005 benchmarks. Each method performs learned macro placement followed by DREAMPlace standard-cell placement. HPWL is reported in 10^6 . Lower is better.

Benchmark	Diffusion-1000 + DP	Diffusion-128 + DP	CD-128 + DP
adaptec1	79.08	94.50	88.65
adaptec2	105.69	121.00	126.36
adaptec3	231.39	207.80	172.56
adaptec4	207.51	205.10	176.29
bigblue1	87.12	80.91	84.26
Average	142.16	141.86	129.62

Table 10 compares the full-chip placement results of the hierarchical approach for diffusion model and CD-128 flows. On average, hierarchical diffusion obtains an HPWL of $30.25(\times 10^6)$, while hierarchical CD achieves $31.27(\times 10^6)$, corresponding to a modest increase of approximately 3.4%. The hierarchical diffusion flow produces lower HPWL on 15 of the 18 benchmarks, whereas hierarchical CD-128 achieves better results on IBM14, IBM15, and IBM17. The relatively small average gap indicates that the distilled model preserves much of the global cluster and macro organization required by the subsequent boundary-aware expansion, cluster-local refinement, full-chip assembly, and legalization stages. The remaining difference suggests that hierarchical placement is sensitive to small variations in predicted cluster positions because inter-cluster connectivity is approximated during independent local optimization. Overall, these results demonstrate that consistency distillation can support a complete hierarchical placement pipeline while providing a favorable tradeoff between placement quality and generative inference cost.

The proposed hierarchical framework is not restricted to a specific global placement model. It can be integrated with other macro-placement methods by representing standard-

cell groups as placeable clusters and converting the resulting reduced circuit into the input format required by the selected placer. The data-preparation scripts, format adapters, execution pipeline, and reproduction instructions are provided in the public GitHub repository accompanying this work.

Table 10. Full detailed placement results of the hierarchical diffusion model and consistency-distilled-128 flows. HPWL is reported in 10^6 . Lower is better.

Benchmark	Hier. Diffusion	Hier. CD
ibm01	2.93	2.98
ibm02	6.02	6.59
ibm03	9.05	10.66
ibm04	9.84	9.86
ibm05	11.84	11.87
ibm06	8.45	8.46
ibm07	13.18	13.58
ibm08	14.52	15.67
ibm09	17.07	18.43
ibm10	38.43	45.82
ibm11	22.84	26.43
ibm12	48.33	48.60
ibm13	30.29	33.20
ibm14	44.25	44.05
ibm15	60.77	60.35
ibm16	70.81	71.74
ibm17	81.20	78.14
ibm18	54.75	56.42
Average	30.25	31.27

3.5. Global-Routing Evaluation

To examine whether the placement-level HPWL improvements translate into downstream physical-design benefits, we performed global routing on the detailed placements of the IBM benchmarks using identical OpenROAD (<https://openroad.org>) settings. The global-routing results in Table 11 show that **CD-128 + DP** provides the most favorable overall downstream routing characteristics among the evaluated placement flows. In particular, CD-128 achieves the lowest routed wirelength of $4.208 \times 10^6 \mu\text{m}$, compared with $4.424 \times 10^6 \mu\text{m}$ for Diffusion-1000 and $4.505 \times 10^6 \mu\text{m}$ for DDIM-128. It also produces the lowest routing usage which is the percentage of available routing resources consumed by routed nets (**100.34%**), via count (726.07×10^3), and routing runtime (**436.78 s**), indicating that the placement-level HPWL improvements translate into shorter interconnects and more efficient routing. Although its normalized overflow (**0.3134**) is slightly higher than that of Diffusion-1000 (**0.3069**) and DDIM-128 (**0.3103**), the difference is small, suggesting that the reduction in wirelength and routing cost is achieved without a substantial degradation in congestion. Overall, these results indicate that the distilled CD-128 placements remain competitive in routability while providing better routed wirelength, fewer vias, and lower routing runtime than the diffusion-based alternatives.

Table 11. Average global-routing comparison over the 18 IBM benchmarks using identical OpenROAD settings. Normalized overflow measures routing congestion. Routed wirelength is reported in $10^6 \mu\text{m}$ and vias in 10^3 . Lower is better.

Method	Normalized Overflow	Usage (%)	Routed WL	Vias	Runtime (s)
Diffusion-1000 + DP	0.3069	105.58	4.424	741.57	473.06
DDIM-128 + DP	0.3103	106.67	4.505	757.65	490.72
CD-128 + DP	0.3134	100.34	4.208	726.07	436.78

3.6. Qualitative Placement Comparison

Figure 3 provides a qualitative comparison of the full-placement layouts generated by the learned placement flows on IBM01. The CD-128 and diffusion model flows use learned macro placement followed by DREAMPlace-based standard-cell refinement, while using them in a hierarchical manner predicts both macro and cluster locations before performing boundary-aware cluster expansion, cluster-local standard-cell optimization, and final full-chip assembly. Presenting both conventional and hierarchical variants illustrates how each generative model influences the spatial organization of macros and standard cells under the two full-placement strategies. These visualizations complement the quantitative HPWL and runtime results by showing how the evaluated flows distribute cells and fixed macros across the chip canvas.



Figure 3. Full-circuit placement visualizations for IBM01 under four learned placement flows. (a) CD-128 generates the macro placement before DREAMPlace performs standard-cell placement. (b) Hierarchical CD-128 places macros and clusters globally, followed by cluster-local standard-cell optimization and full-chip assembly. (c) The diffusion model generates the macro placement before DREAMPlace performs standard-cell placement. (d) Hierarchical diffusion first places macros and clusters globally and then performs cluster expansion, local optimization, and final assembly.

4. Discussion

The results support the main hypothesis of this work: consistency distillation can substantially reduce the inference cost of diffusion-based chip placement while preserving useful placement structure. On the clustered IBM benchmark, the diffusion teacher remains the strongest quality reference among the learned models, but it requires a long iterative sampling process. The consistency-distilled models replace this expensive trajectory with a controllable number of consistency evaluations. In particular, the 128-step model reduces inference time from 170.42 s to 41.10 s while keeping the average HPWL close to the teacher. This confirms that consistency distillation is a practical acceleration mechanism rather than only a model-compression approach.

The inference-step ablation shows a clear quality-runtime tradeoff. One-step generation is extremely fast but too aggressive for high-quality placement. Moving from one to two steps recovers much of the lost HPWL quality, while four to eight steps provide a strong low-latency operating point with high legality and inference-level speedup. Longer samplers, especially 64 and 128 steps, are more appropriate when HPWL is prioritized. This gives the placement flow a useful inference knob: the same distilled model can be used for rapid design-space exploration or for higher-quality candidate generation.

The macro-only results further illustrate the expected behavior of consistency distillation. On IBM and ISPD2005, the CD-128 model is faster than the diffusion baseline but does not fully match its macro HPWL. This should be interpreted as an acceleration-quality tradeoff, not as an unconditional improvement over the teacher. Nevertheless, the detailed-placement results show that macro-only HPWL is not the only predictor of final full-chip quality. After DREAMPlace standard-cell refinement, CD-128 macro initialization achieves a lower average HPWL than the diffusion-teacher macro initialization.

The hierarchical flow demonstrates a second contribution: the learned placement can be extended from macro/cluster prediction to full-chip reconstruction. By combining hMETIS clustering, boundary-aware cluster expansion, cluster-local DREAMPlace optimization, and final assembly, the method produces complete full-chip placements from learned macro/cluster outputs. The hierarchical approach provides a scalable framework in which standard-cell refinement is decomposed into smaller cluster-level problems that can be processed in parallel.

Several limitations remain. First, the ISPD2005 macro-only result is computed on the subset of samples that completed successfully, so broader ISPD evaluation requires improved macro masking for very large circuits. Second, the hierarchical placement flow needs stronger global refinement after cluster assembly to have a better performance. In addition, the consistency model in this study was distilled for 1,000,000 training steps; longer distillation is likely to improve placement quality and further reduce the gap to the diffusion model. Future work should integrate routability-aware objectives, improve boundary modeling for inter-cluster nets, increase the distillation budget, and add a global post-assembly refinement stage.

5. Conclusions

This paper presented CoDPlacer, a consistency-distilled framework for accelerating digital IC placement. By distilling a pretrained Diffusion teacher into a consistency model, the proposed approach reduces the number of required sampling steps and provides a tunable tradeoff between placement quality and inference time. Experiments on clustered IBM benchmarks show that the 128-step consistency model approaches the teacher's HPWL quality while providing substantial generation-level speedups. Macro-only experiments on IBM and ISPD2005 further show that the distilled model can reduce runtime compared with diffusion-based sampling, although some macro-HPWL degradation remains.

The paper also evaluated two full-placement strategies. In the conventional macro-placement flow, CD-128 macro locations are fixed and DREAMPlace performs full standard-cell placement, producing competitive detailed-placement results and improving over diffusion-teacher macro initialization on average. In the hierarchical flow, standard cells are grouped using hMETIS, globally placed as clusters with macros, expanded using boundary-aware constraints, optimized locally with DREAMPlace, and assembled into a full-chip placement which demonstrates a complete learned hierarchical placement pipeline.

Overall, the results indicate that consistency distillation is a promising direction for reducing the inference cost of generative chip placement and serves as a fast learned placement initialization strategy with controllable runtime-quality behavior.

Author Contributions: Conceptualization, M.A.S.; methodology, M.A.S.; software, M.A.S.; validation, M.A.S.; formal analysis, M.A.S.; investigation, M.A.S.; resources, J.S.Y.; data curation, M.A.S.; writing—original draft preparation, M.A.S.; writing—review and editing, M.A.S. and J.S.Y.; visualization, M.A.S.; supervision, J.S.Y.; project administration, J.S.Y.; funding acquisition, J.S.Y. and M.A.S. All authors have read and agreed to the published version of the manuscript.

Funding: This work is supported in part by Tokyo Electron America and Florida High Tech Corridor matching grants.

Data Availability Statement: The IBM [36] and ISPD [37] benchmark datasets analyzed in this study are publicly available. The source code, experiment configurations, data-preparation scripts, and evaluation procedures developed for this study are available at <https://github.com/mohammad-AJP/CoDPlacer>, accessed on 12 August 2026.

Conflicts of Interest: The authors declare no conflicts of interest.

Appendix A. Ablation Study

Per-Circuit HPWL Analysis

Table A1 reports the per-circuit HPWL. The one-step consistency model is consistently worse than the other settings, which agrees with the aggregate result in Table 4. From two steps onward, the model becomes competitive across all IBM circuits, and the longer 64-step and 128-step samplers usually provide the best consistency-model HPWL.

Table A1. Per-circuit average HPWL for the clustered IBM comparison over 5 seeds. HPWL is reported in 10^6 . Lower is better.

Circuit	Reference	Teacher	CD-1	CD-2	CD-4	CD-8	CD-16	CD-64	CD-128
IBM01	2.528	2.760	8.278	3.644	3.354	3.212	2.919	2.752	2.735
IBM02	5.239	6.040	17.635	8.953	8.017	8.086	7.666	7.110	6.830
IBM03	8.551	8.258	22.599	12.477	10.318	9.980	9.990	9.835	8.848
IBM04	8.676	9.541	21.056	12.186	10.625	10.353	10.127	9.787	9.401
IBM05	9.622	10.628	25.148	13.053	12.138	12.177	11.447	10.809	10.584
IBM06	7.547	8.229	20.203	10.856	9.965	9.531	9.515	8.574	8.249
IBM07	12.169	12.755	37.565	16.685	15.379	14.839	15.087	13.744	13.461
IBM08	14.411	14.282	33.581	17.501	17.338	17.887	17.408	15.449	15.522
IBM09	13.240	14.989	40.984	19.660	17.956	16.730	17.280	16.417	14.377
IBM10	36.514	36.969	112.899	51.649	48.696	51.685	49.506	46.604	42.599
IBM11	20.311	20.505	61.520	29.149	24.765	25.013	24.628	22.137	21.627
IBM12	42.407	45.285	116.211	62.242	54.956	58.264	53.788	49.537	45.210
IBM13	24.344	26.110	78.646	35.949	31.409	31.112	30.291	28.178	26.222
IBM14	40.163	41.895	130.218	58.681	49.888	49.646	47.554	45.058	43.490
IBM15	50.197	54.590	154.277	72.081	62.011	59.039	57.144	57.807	52.666
IBM16	66.210	66.071	204.198	89.490	82.974	83.556	81.035	75.581	69.072
IBM17	78.735	76.881	235.291	112.194	98.447	92.499	95.731	84.790	80.454
IBM18	46.187	51.070	164.482	63.227	58.962	56.689	56.840	54.591	50.172
Average	27.058	28.159	82.488	38.315	34.289	33.905	33.220	31.042	28.973

References

- Cheng, C.-K.; Kahng, A.B.; Kang, I.; Wang, L. RePlAce: Advancing Solution Quality and Routability Validation in Global Placement. *IEEE Trans. Comput.-Aided Des. Integr. Circuits Syst.* **2019**, *38*, 1717–1730. [\[CrossRef\]](#)
- Lu, J.; Chen, P.; Chang, C.-C.; Sha, L.; Huang, D.J.-H.; Teng, C.-C.; Cheng, C.-K. ePlace: Electrostatics-Based Placement Using Fast Fourier Transform and Nesterov's Method. *ACM Trans. Des. Autom. Electron. Syst.* **2015**, *20*, 1–34.
- Lin, Y.; Dhar, S.; Li, W.; Ren, H.; Khailany, B.; Pan, D.Z. DREAMPlace: Deep Learning Toolkit-Enabled GPU Acceleration for Modern VLSI Placement. In Proceedings of the 56th Annual Design Automation Conference (DAC), Las Vegas, NV, USA, 2–6 June 2019; pp. 1–6. [\[CrossRef\]](#)
- Kahng, A.B.; Wang, Z. DG-RePlAce: A Dataflow-Driven GPU-Accelerated Analytical Global Placement Framework for Machine Learning Accelerators. *IEEE Trans. Comput.-Aided Des. Integr. Circuits Syst.* **2025**, *44*, 696–708. [\[CrossRef\]](#)
- Lin, Y.; Li, W.; Gu, J.; Ren, H.; Khailany, B.; Pan, D.Z. ABCDPlace: Accelerated Batch-Based Concurrent Detailed Placement on Multithreaded CPUs and GPUs. *IEEE Trans. Comput.-Aided Des. Integr. Circuits Syst.* **2020**, *39*, 5083–5096. [\[CrossRef\]](#)
- Liu, L.; Fu, B.; Lin, S.; Liu, J.; Young, E.F.Y.; Wong, M.D.F. Xplace: An Extremely Fast and Extensible Placement Framework. *IEEE Trans. Comput.-Aided Des. Integr. Circuits Syst.* **2024**, *43*, 1872–1885. [\[CrossRef\]](#)
- Chen, Y.; Wen, Z.; Liang, Y.; Lin, Y. Stronger Mixed-Size Placement Backbone Considering Second-Order Information. In Proceedings of the IEEE/ACM International Conference on Computer-Aided Design (ICCAD), San Francisco, CA, USA, 28 October–2 November 2023.
- Hao, R.; Cai, Y.; Zhou, Q.; Wang, R. DrPlace: A Deep Learning Based Routability-Driven VLSI Placement Algorithm. *J. Comput.-Aided Des. Comput. Graph.* **2021**, *33*, 624–631. [\[CrossRef\]](#)
- Agnesina, A.; Liang, R.; Pradipta, G.; Rajaram, A.; Ren, H. GOALPlace: Begin with the End in Mind. In Proceedings of the 2025 International Symposium on Physical Design (ISPD), Austin, TX, USA, 16–19 March 2025; pp. 2–10. [\[CrossRef\]](#)
- Duran, C.; Roa, E. Routing-Aware Standard Cell Placement Algorithm Applying Boolean Satisfiability. In Proceedings of the IEEE International Symposium on Circuits and Systems (ISCAS), Daegu, Republic of Korea, 22–28 May 2021.
- Li, G.; Yang, Y.; Li, S.; Qu, X.; Lyu, N.; Li, S.E. Decision Making of Autonomous Vehicles in Lane Change Scenarios: Deep Reinforcement Learning Approaches with Risk Awareness. *Transp. Res. Part C Emerg. Technol.* **2022**, *134*, 103452. [\[CrossRef\]](#)
- Mousavi, A.; Mousavi, R.; Mousavi, Y.; Fekih, A. Event-Triggered Reinforcement Learning-Based Fractional-Order Sliding Mode Control with Neural Approximation for Floating Offshore Wind Turbines. *Energy* **2026**, *360*, 100076. [\[CrossRef\]](#)
- Alijanpour, M.; Raie, A. Video Event Recognition Using Two-Stream Convolutional Neural Networks. In Proceedings of the 5th International Conference on Pattern Recognition and Image Analysis (IPRIA), Kashan, Iran, 28–29 April 2021; pp. 1–5. [\[CrossRef\]](#)
- Wu, Y.; Chen, Y.; Yuan, L.; Liu, Z.; Wang, L.; Li, H.; Fu, Y.R. Rethinking Classification and Localization for Object Detection. In Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR), Virtual, 13–19 June 2020.
- Cheng, R.; Yan, J. On Joint Learning for Solving Placement and Routing in Chip Design. In Proceedings of the Conference on Advances in Neural Information Processing Systems (NeurIPS), Virtual, 6–14 December 2021.
- Cheng, C.-K.; Kahng, A.B.; Kundu, S.; Wang, Y.; Wang, Z. Assessment of Reinforcement Learning for Macro Placement. In Proceedings of the International Symposium on Physical Design (ISPD), Virtual, 26–29 March 2023.
- Markov, I.L. Reevaluating Google's Reinforcement Learning for IC Macro Placement. *Commun. ACM* **2024**, *67*, 60–71. [\[CrossRef\]](#)
- Geng, Z.; Wang, J.; Liu, Z.; Xu, S.; Tang, Z.; Yuan, M.; Hao, J.; Zhang, Y.; Wu, F. Reinforcement Learning within Tree Search for Fast Macro Placement. In Proceedings of the 41st International Conference on Machine Learning (ICML), Vienna, Austria, 21–27 July 2024.
- Xue, K.; Chen, R.-T.; Lin, X.; Shi, Y.; Kai, S.; Xu, S.; Qian, C. Reinforcement Learning Policy as Macro Regulator Rather than Macro Placer. In Proceedings of the 38th International Conference on Neural Information Processing Systems (NIPS '24), Vancouver, BC, Canada, 10–15 December 2024; Volume 37, pp. 140565–140588. [\[CrossRef\]](#)
- Agnesina, A.; Rajvanshi, P.; Yang, T.; Pradipta, G.; Jiao, A.; Keller, B.; Khailany, B.; Ren, H. AutoDMP: Automated DREAMPlace-Based Macro Placement. In Proceedings of the 2023 International Symposium on Physical Design (ISPD), Virtual, 26–29 March 2023; pp. 149–157. [\[CrossRef\]](#)
- Hou, Y.; Ye, H.; Zhang, Y.; Xu, S.; Song, G. RoutePlacer: An End-to-End Routability-Aware Placer with Graph Neural Network. In Proceedings of the 30th ACM SIGKDD Conference on Knowledge Discovery and Data Mining (KDD), Barcelona, Spain, 25–29 August 2024; pp. 1085–1095. [\[CrossRef\]](#)
- Lai, Y.; Mu, Y.; Luo, P. MaskPlace: Fast Chip Placement via Reinforced Visual Representation Learning. In Proceedings of the Conference on Advances in Neural Information Processing Systems (NeurIPS), New Orleans, LA, USA, 28 November–9 December 2022.
- Lai, Y.; Liu, J.; Tang, Z.; Wang, B.; Hao, J.; Luo, P. ChiPFormer: Transferable Chip Placement via Offline Decision Transformer. In Proceedings of the 40th International Conference on Machine Learning (ICML), Honolulu, HI, USA, 23–29 July 2023.

24. Cheng, R.; Lyu, X.; Li, Y.; Ye, J.; Hao, J.; Yan, J. The Policy-Gradient Placement and Generative Routing Neural Networks for Chip Design. In Proceedings of the Conference on Advances in Neural Information Processing Systems (NeurIPS), New Orleans, LA, USA, 28 November–9 December 2022.
25. Ying, R.; He, R.; Chen, K.; Eksombatchai, P.; Hamilton, W.L.; Leskovec, J. Graph Convolutional Neural Networks for Web-Scale Recommender Systems. In Proceedings of the 24th ACM SIGKDD International Conference on Knowledge Discovery & Data Mining, London, UK, 19–23 August 2018.
26. Mousa, R.; Sarabadani, A.; Taami, T.; Bengari, A.A.; Eslamifar, O.; Alijanpour Shalmani, M.; Karimi Shahmarvandi, E. A Comparative Survey on Large Language Models for Biological Data. *Preprints* **2025**. [[CrossRef](#)]
27. Lu, Y.-C.; Lim, S. K. VLSI Placement Optimization Using Graph Neural Networks. In Proceedings of the NeurIPS Workshop on Machine Learning for Systems, Virtual, 11 December 2020.
28. Lu, Y.-C.; Pentapati, S.; Lim, S. K. The Law of Attraction: Affinity-Aware Placement Optimization Using Graph Neural Networks. In Proceedings of the International Symposium on Physical Design (ISPD), Virtual, 22–24 March 2021.
29. Hou, Y.; Ye, H.; Yang, S.; Zhang, Y.; Xu, S.; Song, G. TransPlace: Transferable Circuit Global Placement via Graph Neural Network. In Proceedings of the 31st ACM SIGKDD Conference on Knowledge Discovery and Data Mining (KDD), Toronto, ON, Canada, 3–7 August 2025; pp. 484–495. [[CrossRef](#)]
30. Shahane, A.; Manjiri, S.S.; Jain, A.; Kumar, S. Graph of Circuits with GNN for Exploring the Optimal Design Space. In Proceedings of the 37th International Conference on Neural Information Processing Systems (NIPS '23), New Orleans, LA, USA, 10–16 December 2023; Volume 36, pp. 6014–6025. [[CrossRef](#)]
31. Lu, Y.-C.; Yang, T.; Lim, S.K.; Ren, H. Placement Optimization via PPA-Directed Graph Clustering. In Proceedings of the ACM/IEEE Workshop on Machine Learning for CAD, Snowbird, Utah, USA, 12–13 September 2022.
32. Liu, Y.; Zhou, H.; Wang, J.; Yang, F.; Zeng, X.; Shang, L. The Power of Graph Signal Processing for Chip Placement Acceleration. In Proceedings of the 43rd IEEE/ACM International Conference on Computer-Aided Design (ICCAD '24), Newark, NJ, USA, 27–31 October 2024; Association for Computing Machinery: New York, NY, USA, 2025; pp. 1–8. [[CrossRef](#)]
33. Lu, Y.-C.; Ren, H.; Hsiao, H.; Lim, S.K. DREAM-GAN: Advancing DREAMPlace towards Commercial-Quality Using Generative Adversarial Learning. In Proceedings of the International Symposium on Physical Design (ISPD), Virtual, 26–29 March 2023.
34. Lee, V.; Nguyen, M.; Elzeiny, L.; Deng, C.; Abbeel, P.; Wawrzynek, J. Chip Placement with Diffusion Models. In Proceedings of the 42nd International Conference on Machine Learning (ICML), Vancouver, BC, Canada, 13–19 July 2025; Volume 267, pp. 33499–33514.
35. Song, Y.; Dhariwal, P.; Chen, M.; Sutskever, I. Consistency Models. In Proceedings of the 40th International Conference on Machine Learning (ICML), Honolulu, HI, USA, 23–29 July 2023.
36. Adya, S.N.; Chaturvedi, S.; Roy, J.A.; Papa, D.A.; Markov, I.L. Unification of Partitioning, Placement and Floorplanning. In Proceedings of the IEEE/ACM International Conference on Computer-Aided Design (ICCAD), San Jose, CA, USA, 7–11 November 2004; pp. 550–557.
37. Nam, G.-J.; Alpert, C.J.; Villarrubia, P.; Winter, B.; Yildiz, M. The ISPD2005 Placement Contest and Benchmark Suite. In Proceedings of the International Symposium on Physical Design (ISPD), San Francisco, CA, USA, 3–6 April 2005; pp. 216–220. [[CrossRef](#)]
38. Schlag, S.; Heuer, T.; Gottesbüren, L.; Akhremtsev, Y.; Schulz, C.; Sanders, P. High-Quality Hypergraph Partitioning. *ACM J. Exp. Algorithmics* **2023**, *27*, 1–39. [[CrossRef](#)]
39. Karypis, G.; Aggarwal, R.; Kumar, V.; Shekhar, S. Multilevel Hypergraph Partitioning: Applications in VLSI Domain. *IEEE Trans. Very Large Scale Integr. (Vlsi) Syst.* **1999**, *7*, 69–79. [[CrossRef](#)]

Disclaimer/Publisher's Note: The statements, opinions and data contained in all publications are solely those of the individual author(s) and contributor(s) and not of MDPI and/or the editor(s). MDPI and/or the editor(s) disclaim responsibility for any injury to people or property resulting from any ideas, methods, instructions or products referred to in the content.