PPAPlace: Differentiable Cross-Stage Objectives for Chip Placement Optimization

Ruogu Chen
University of Alberta
Department of Electrical and Computer Engineering
Edmonton, AB, Canada
ruogu@ualberta.ca

Jie Han
University of Alberta
Department of Electrical and Computer Engineering
Edmonton, AB, Canada
jhan8@ualberta.ca

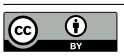
Abstract

Macro placement significantly affects a chip's post-route performance, power, and area (PPA). Most placement methods optimize half-perimeter wirelength (HPWL) as the primary objective. However, recent benchmarking shows a near-zero correlation between HPWL and post-route timing metrics such as the worst negative slack (WNS) and total negative slack (TNS). As a result, all six evaluated artificial intelligence (AI) placers degraded PPA relative to the hierarchical baseline. Recent efforts have tried to train cross-stage predictors to close this gap. However, existing methods focus on macro-only representations and use pre-route metrics as training labels. A label fidelity study of ten circuits at four design flow stages reveals that HPWL and pre-route timing poorly reflect final post-route timing rankings. In contrast, post-global-routing achieves the best balance between final timing fidelity and label generation cost-effectiveness. Based on this finding, PPAPlace is a timing-driven differentiable surrogate predicting post-route PPA from macro and standard-cell placements. The surrogate is a dual-stream predictor that combines graph attention over the chip netlist with spatial convolution over the placement grid. It is trained on post-global-routing labels. The predicted WNS and TNS gradients flow end-to-end back to cell coordinates. PPAPlace exploits these gradients in two ways: as a co-objective injected into an analytical placer's optimization loop (PPAPlace-CoOpt), and as a post-placement refinement step that adjusts macro positions via projected gradient descent (PPAPlace-Refine). On five ChiPBench test circuits excluded from training, PPAPlace improves average WNS and TNS by 22% and 51% over the hierarchical baseline while preserving power and routability, using the same predictor without test-circuit retraining. Code is available at https://github.com/ValleyC/PPAPlace.

Keywords

chip placement, PPA prediction, differentiable objectives, surrogate-guided optimization, graph attention network, physical design

ACM Reference Format:
Ruogu Chen and Jie Han. 2026. PPAPlace: Differentiable Cross-Stage Objectives for Chip Placement Optimization. In *IEEE/ACM International Conference on Computer-Aided Design (ICCAD '26), November 08–12, 2026, San Jose, CA, USA*. ACM, New York, NY, USA, 9 pages. https://doi.org/10.1145/3831252.3834172



This work is licensed under a Creative Commons Attribution 4.0 International License.
ICCAD '26, San Jose, CA, USA
© 2026 Copyright held by the owner/author(s).
ACM ISBN 979-8-4007-2873-0/2026/11
https://doi.org/10.1145/3831252.3834172

1 Introduction

Chip placement determines the physical locations of circuit modules on a two-dimensional canvas and is one of the most important steps in very-large-scale integration physical design. The quality of placement directly affects a chip's performance, power consumption, and area (PPA), which together determine whether a design meets its specifications for manufacturing [10].

Existing placement approaches fall into two broad categories: analytical and artificial-intelligence (AI)-based placers. Analytical placers such as DREAMPlace [16] formulate placement as continuous optimization of a smooth wirelength approximation with density penalties, solved by gradient descent on a graphics processing unit (GPU). AI-based methods, including reinforcement learning (RL) [12, 13, 19], black-box optimization [22], and diffusion models [14], replace or augment the optimizer with learned policies. Despite the methodological differences between these two categories, most methods optimize the half-perimeter wirelength (HPWL), a computationally inexpensive estimate of the total wire length, as the primary objective.

However, growing evidence suggests that HPWL is not as reliable a proxy as commonly assumed. The ChiPBench benchmark [26] evaluated six AI-based placement methods on 20 circuits through the full OpenROAD [2] chip design flow. It uses Hier-RTLMP (Hierarchical Register-Transfer-Level Macro Placer) [11], OpenROAD's built-in macro placer, as the baseline placement method. Hier-RTLMP leverages the RTL design hierarchy and dataflow structure rather than optimizing wirelength alone. The results show that every evaluated AI method in ChiPBench degraded PPA relative to Hier-RTLMP. ChiPBench further reported a Pearson correlation of only -0.08 between macro HPWL and the worst negative slack (WNS), indicating that wirelength optimization is essentially uninformative for timing.

Recent efforts have begun to close this gap from two directions. AutoDMP [1] tunes DREAMPlace's configuration parameters through multi-objective Bayesian optimization (BO). It uses post-placement proxies such as rectilinear Steiner minimum tree (RSMT) wirelength, cell density, and rectangular uniform wire density (RUDY) congestion to guide the search. While effective at improving placement diversity, these proxies are computed before routing and suffer a similar proxy-to-PPA gap. Therefore, they do not always stay faithful to the post-route PPA. LaMPlace [6] takes a complementary approach. It trains a cross-stage predictor on offline placement data and uses it to guide macro placement through evolutionary search, achieving notable timing improvements. While these methods show that cross-stage prediction is

a viable path to better placement, they share two unexamined assumptions. First, they use pre-route static timing analysis (STA) as supervision without verifying that these labels faithfully preserve post-route PPA rankings. Second, existing methods represent the placement through macro positions alone, discarding the standard cell density and routing congestion that ultimately govern timing and power.

This paper examines both assumptions. A controlled study across ten ChiPBench circuits reveals that pre-route timing labels, used as supervision data in prior works, can be misleading. Post-global-routing (GRT) labels, by contrast, consistently achieve high fidelity with final post-route timing and remain cost-effective in generation. This finding establishes a principled basis for selecting the supervision stage in any future cross-stage learning approach.

Built on this finding, PPAPlace uses post-GRT supervision to construct a differentiable post-route timing objective from the complete placement state of macros and standard cells. A dual-stream predictor is trained on post-GRT labels. The graph attention stream encodes netlist connectivity. The spatial convolution stream encodes cell density, pin density, and RUDY-style congestion. Every component operates within a differentiable computation graph, enabling end-to-end gradient flow from predicted timing back to cell coordinates.

Because the surrogate is fully differentiable, it provides not only PPA predictions but also timing gradients that indicate how moving each cell would affect post-route timing. These gradients can be injected directly into a differentiable placer's optimization loop as a learned co-objective, complementing wirelength and density with routing-aware timing feedback. The same predictor also supports post-placement gradient descent to locally refine any converged placement, requiring only a completed legal placement as input.

The novel contributions are as follows:

(1) A label fidelity study across ten circuits and four design stages, establishing post-global-routing as the most cost-effective supervision stage for cross-stage PPA prediction.
(2) A differentiable dual-stream predictor over the complete mixed-size placement state, with end-to-end gradient flow from predicted timing back to cell positions. With the architecture fixed, post-GRT supervision raises Kendall's τ from 0.13 with pre-route STA to 0.31, while the combined setting is nearly 4× the macro-only, pre-route result of 0.08.
(3) Two gradient-guided deployment modes. PPAPlace-CoOpt injects the surrogate as a co-objective into DREAMPlace's analytical loop for global topology guidance. PPAPlace-Refine applies post-placement gradient descent to any converged placement without access to the source placer's internals. Combined, they improve WNS by 22% and total negative slack (TNS) by 51% over the hierarchical baseline, outperforming all evaluated prior methods.

2 Related Work

Analytical placement. Analytical placers formulate placement as continuous optimization of a smooth wirelength objective subject to density constraints [5, 17]. DREAMPlace [16] recast this formulation as a neural network training problem, achieving over 30× GPU speedup. DREAMPlace 4.0 [15] added timing-driven net weighting from pre-route STA. AutoDMP [1] tunes 16 DREAMPlace parameters via multi-objective Tree-structured Parzen Estimator using post-placement proxies (RSMT wirelength, cell density, RUDY congestion) as BO objectives.

Differentiable placement objectives. Recent work has extended gradient-based placement beyond wirelength and density. Efficient-TDP [21] injects pin-to-pin attraction on critical paths into DREAMPlace using pre-route STA, achieving state-of-the-art timing-driven standard-cell placement. RoutePlacer [8] trains a graph neural network on global-router overflow labels and injects the learned congestion penalty as a differentiable objective into DREAMPlace's loop, reducing routing overflow by up to 16%. These methods target a single intermediate metric such as pre-route timing or routability. None addresses mixed-size placement or post-route PPA as an optimization objective.

AI-based placement. AlphaChip [19] pioneered deep reinforcement learning for macro placement, though its reproducibility remains debated [18]. Subsequent work explored visual representation learning [13], offline RL [12], evolutionary search [22], placement refinement [27], and tree-search-guided RL [7]. All optimize HPWL or macro HPWL as the primary objective.

Cross-stage PPA prediction and optimization. LaMPlace [6] trains a Laurent polynomial predictor to estimate cross-stage metrics from macro pairwise distances and uses it to generate an L-mask for sequential greedy macro placement. Its main timing labels come from OpenTimer after standard-cell placement, before CTS or routing. MacroRank [4] ranks macro placements by final routing quality, while PreRoutGNN [29] predicts pre-routing timing at standard-cell placement. Neither integrates its predictor into mixed-size placement optimization. Re^{2}MaP [20] achieves state-of-the-art macro placement through recursive prototyping and packing-tree relocation with hand-engineered cost terms. It represents the best of algorithmic macro placement but does not use learned objectives. BBOPlace-Bench [28] benchmarks black-box optimization approaches to macro placement. It argues for aligning the search objective with downstream PPA. In the commercial space, Synopsys DSO.ai [24] and Cadence Cerebrus [3] use reinforcement learning to tune tool configurations across the design flow, treating placement as part of a broader design-space optimization problem. Across these efforts, no work has systematically validated which design flow stage provides the most reliable supervision for cross-stage prediction.

3 Preliminaries

3.1 Chip Placement and Evaluation Metrics

Chip placement assigns physical positions to circuit modules on a two-dimensional canvas. A netlist hypergraph $H = (V, E)$ specifies the modules V (macros and standard cells) and the nets E connecting them. A placement method seeks positions $\mathbf{x} = \{(x_i, y_i)\}_{i=1}^{N}$ that minimize a placement objective. In analytical placers such as DREAMPlace [16], this takes the form:

$$\min_{\mathbf{x}} \ \mathcal{L}(\mathbf{x}) = \mathcal{W}(\mathbf{x}) + \lambda \cdot \mathcal{D}(\mathbf{x}), \tag{1}$$

where $\mathcal{W}$ is a smooth wirelength approximation and $\mathcal{D}$ is a density penalty that discourages cell overlap [17]. The density weight λ

Table 1: Spearman ρ_{WNS} between intermediate and post-route timing across ten ChiPBench circuits (macro count in parentheses). Green : $\rho \geq 0.7$; light : $0.3 \leq \rho < 0.7$; pink : negative. ρ_{TNS} follows the same pattern. Rightmost column: average wall-clock cost of generating one label at each stage, shown on a blue scale to distinguish it from correlation. CTS: clock-tree synthesis; DRT: detailed routing. Post-DRT is the correlation reference ($\rho \equiv 1$, 3.7 hrs on average).

Stage	bp_be (10)	bp_fe (11)	bp_be12 (12)	isa_npu (15)	swerv (28)	vga_lcd (62)	ether (64)	dft68 (68)	mor1kx (78)	ari133 (132)	**Avg**	**Time** (hrs)
HPWL	+.12	−.15	+.08	−.21	+.05	−.18	−.09	+.14	−.11	+.03	−.03	<0.01
Pre-CTS STA	+.15	+.09	−.07	+.18	−.11	+.06	−.14	+.22	+.03	+.12	+.05	0.08
Post-CTS	.83	.64	.71	.38	−.12	.25	−.31	.47	.19	.77	.38	0.14
Post-GRT	**.80**	**.91**	**.87**	**.85**	**.89**	**.82**	**.78**	**.84**	**.86**	**.94**	**.86**	**0.20**

increases iteratively to enforce legality. The wirelength term is typically based on HPWL:

$$\text{HPWL}(\mathbf{x}) = \sum_{e \in E} \left[\max_{i \in e} x_i - \min_{i \in e} x_i + \max_{i \in e} y_i - \min_{i \in e} y_i \right], \quad (2)$$

which estimates the total wire length by summing the bounding-box half-perimeters across all nets. HPWL is differentiable (via smooth approximations such as the weighted-average model [9]), decomposable across nets, and fast to compute.

However, the actual quality of a placed design is determined by post-route PPA metrics obtained after executing the downstream flow: clock tree synthesis (CTS), global routing (GRT), and detailed routing (DRT), each adding fidelity to timing and congestion estimates. Let $F(\mathbf{x})$ denote the full downstream flow and $F_s(\mathbf{x})$ the same flow ending at stage s. Their PPA metric outputs, used as labels, are

$$\mathbf{y} = F(\mathbf{x}) = [\text{WNS},\ \text{TNS},\ \text{Power},\ \text{Area}], \quad (3)$$

$$\mathbf{y}_s = F_s(\mathbf{x}), \quad s \in \{\text{CTS},\ \text{GRT},\ \text{DRT}\}, \quad (4)$$

where WNS is the worst negative slack, TNS is the total negative slack, and power and area are the total power consumption and physical footprint. Computing $\mathbf{y}$ typically takes tens of minutes to hours.

A cross-stage PPA predictor f_θ, parameterized by θ, approximates $\mathbf{y}_{\text{DRT}}$ from placement-stage features, thus avoiding the cost of running $F(\mathbf{x})$ at inference time. A prerequisite is that the training labels $\mathbf{y}_s$ used to supervise f_θ must rank placements of different quality consistently with the final outcome $\mathbf{y}_{\text{DRT}}$. Section 4 investigates which stage provides labels that best preserve the final post-route ranking.

4 Label Fidelity Analysis

4.1 Setup

This study spans 10 ChiPBench [26] circuits covering RISC-V CPUs from three families (bp_fe, bp_be, and bp_be12 from BlackParrot, swerv_wrapper from SweRV, ariane133 from Ariane), an OpenRISC CPU (mor1kx), a neural processing unit (isa_npu), and peripheral designs (ethernet, dft68, vga_lcd). The number of macros ranges from 10 to 132 and cell counts from 33K to 427K, ensuring that the findings are not specific to a single design scale or application domain. Note that this fidelity test uses RTLMP weight configurations evaluated through the full flow, and is independent of the training/test split used for the main experiments in Section 6.

To generate diverse placements for each circuit, the weight parameters of OpenROAD's RTLMP hierarchical macro placer [11] are systematically varied. RTLMP optimizes a weighted combination of six objectives controlled by AREA_WT, WIRELENGTH_WT, BOUNDARY_WT, OUTLINE_WT, NOTCH_WT, and DEAD_SPACE. For each circuit, 20 configurations are evaluated: 1 default (all weights at their ChiPBench defaults), 12 single-parameter sweeps (each of the six weights set to a high value of 20 or a low value of 0.1, with all others at default), and 7 multi-parameter combinations that probe pairwise interactions (e.g., area+wirelength both at 10, wirelength+boundary both at 10) and global extremes (all three main weights at 10 or at 0.1). RTLMP is deterministic, so the 20 settings form a controlled sweep rather than random placement samples, yielding 200 placements across the 10 circuits.

Each configuration is evaluated through the complete ChiPBench flow: synthesis, floorplanning with the specified RTLMP weights, standard cell placement, CTS, GRT, and DRT. PPA metrics (WNS, TNS, power) are recorded at each stage. Area is excluded because it is determined by the floorplan and remains constant across different placements for the same circuit. The single ChiPBench pipeline uses the same synthesized netlist and flow settings across stages, with stage-specific parasitic estimates. Post-DRT metrics serve as the ground truth throughout this analysis.

4.2 Stage-Wise Rank Correlation

Table 1 reports Spearman ρ_{WNS} between each intermediate stage and the post-route ground truth. WNS and TNS show the same pattern. Only WNS is reported.

HPWL has near-zero average correlation ($\bar{\rho} = -0.03$), with negative values on 5 of 10 circuits. Pre-CTS STA, the label source used by LaMPlace [6], is similarly uninformative ($\bar{\rho} = +0.05$). Post-CTS is inconsistent: strongly positive on some circuits (bp_be: $\rho = 0.83$) but negative on others (ethernet: $\rho = -0.31$). Post-GRT achieves $\rho \geq 0.78$ on all 10 circuits with no sign reversals ($\bar{\rho} = 0.86$). It costs 0.20 hrs per sample on average versus 3.7 hrs for full DRT (Table 1). It is thus the most cost-effective supervision stage.

5 PPAPlace

Based on the findings in Section 4, PPAPlace uses post-global-routing labels as training supervision. PPAPlace has three components. A differentiable PPA predictor is trained on the mixed-size placement state (Sections 5.1–5.3). A differentiable feature extraction layer enables gradient flow from predictions back to cell

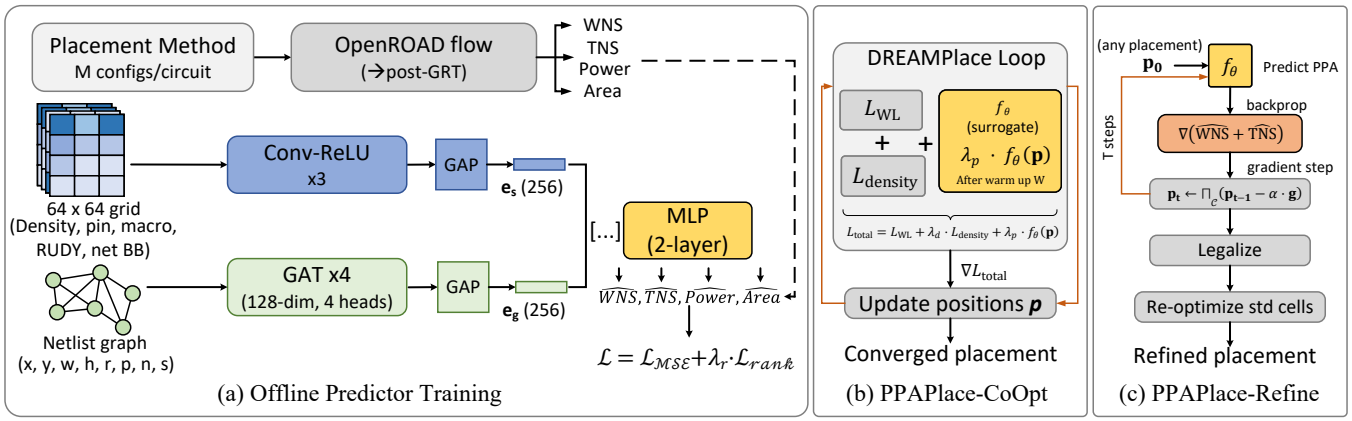


Figure 1: PPAPlace framework. (a) Offline training on post-GRT labels. (b) CoOpt: surrogate gradients injected into DREAMPlace as a co-objective. (c) Refine: post-placement gradient descent on the converged result.

positions (Section 5.4). Two gradient-guided deployment modes, co-objective placement and post-placement refinement, exploit these gradients to improve placement quality (Section 5.5). Figure 1 illustrates the overall design.

5.1 Mixed-Size Placement Representation

Existing cross-stage predictors such as LaMPlace [6] represent the placement state through macro pairwise distances, discarding information about standard cells, spatial density, and routing congestion. However, timing violations and power consumption are largely affected by standard cell placement and routing, not macro positions alone [26]. PPAPlace addresses this by observing the complete mixed-size placement through two complementary representations.

Spatial representation. The placement canvas is rasterized into a 64×64 grid with five channels:

(1) Cell density: total cell area overlapping each bin, normalized by bin capacity.
(2) Pin density: I/O pin concentration per bin.
(3) Macro occupancy: degree of macro presence in each bin.
(4) RUDY-style proxy [23]: the sum of inverse net bounding-box areas, $\sum_e (W_e H_e)^{-1}$, for boxes overlapping the bin. Here, W_e and H_e are the box dimensions.
(5) Net bounding box density: net bounding box overlap per bin, capturing routing pressure from nets whose pins lie outside the bin.

All channels are normalized to $[0, 1]$. This grid-based representation captures spatial distribution patterns invisible to macro-only predictors, including congestion hotspots and density imbalances.

Graph representation. The netlist is represented as a graph $G = (V_G, E_G)$ where macros are the nodes. Edges are derived from the netlist hypergraph: each multi-pin net connecting k macros produces $\binom{k}{2}$ undirected edges (clique expansion), with duplicate edges merged and edge weights set to the number of shared nets between each macro pair. Each node carries a feature vector:

$$\mathbf{h}_i = [x_i,\ y_i,\ w_i,\ h_i,\ r_i,\ p_i,\ n_i,\ s_i], \tag{5}$$

where (x_i, y_i) denotes the normalized position, (w_i, h_i) the normalized width and height, $r_i = w_i/h_i$ the aspect ratio, p_i the pin count, n_i the net degree (number of nets connected to this node), and s_i the average net span (mean HPWL of connected nets).

The two streams are complementary: the spatial grid captures global density and routing pressure while the graph captures per-macro identity and local connectivity.

5.2 Dual-Stream PPA Predictor

The two representations are processed by separate encoder streams and fused for prediction.

Spatial stream. A lightweight convolutional neural network (CNN) with 3 layers and ReLU activations progressively reduces the spatial resolution of the 64×64 grid. Global average pooling (GAP) produces a spatial embedding $\mathbf{e}_s \in \mathbb{R}^{256}$.

Graph stream. A 4-layer graph attention network (GAT) [25] with 128-dimensional hidden states and 4 attention heads processes the netlist graph. Multi-head attention allows each layer to learn different types of relationships between connected nodes, such as spatial proximity and connectivity strength. GAP aggregates all node embeddings into a fixed-size graph embedding $\mathbf{e}_g \in \mathbb{R}^{256}$, regardless of circuit size.

Fusion and prediction. The two embeddings are concatenated and mapped to PPA predictions through a 2-layer multi-layer perceptron (MLP):

$$f_\theta(\mathbf{x}) = \mathrm{MLP}\big([\mathbf{e}_g(\mathbf{x});\ \mathbf{e}_s(\mathbf{x})]\big) \to [\widehat{\mathrm{WNS}},\ \widehat{\mathrm{TNS}},\ \widehat{\mathrm{Power}},\ \widehat{\mathrm{Area}}]. \tag{6}$$

The hat notation denotes predicted values. The architecture produces fixed-size embeddings (512 dimensions in total) regardless of circuit size, enabling the same trained model to be applied across circuits with different numbers of cells and macros. Because the entire pipeline (feature extraction, GAT, CNN, MLP) is composed of differentiable operations, the Jacobian $\partial f_\theta / \partial \mathbf{p}$, where $\mathbf{p}$ is the vector of cell positions, is available via automatic differentiation. This Jacobian indicates how each cell's position influences predicted post-route PPA.

5.3 Training

5.3.1 Data Generation. Training data are generated by running a placement method with M randomized configurations per circuit. Each configuration varies parameters that affect placement quality, such as target density and density weight schedule. As discussed in Section 4, each resulting placement is evaluated through the OpenROAD flow to post-global-routing, producing labels $\mathbf{y}_i = [\text{WNS}, \text{TNS}, \text{Power}, \text{Area}]$. The training set is $\mathcal{D} = \{(\mathbf{x}_i, \mathbf{y}_i)\}_{i=1}^{C\times M}$ across C circuits.

5.3.2 Loss Function. The predictor is trained with a composite loss:

$$\mathcal{L} = \mathcal{L}_{\text{MSE}} + \lambda_r \cdot \mathcal{L}_{\text{rank}}. \tag{7}$$

$\mathcal{L}_{\text{MSE}}$ is the mean squared error between predicted and true PPA values. However, the core task of the predictor is not to estimate exact PPA values, but to correctly rank which placements produce superior PPA and which produce inferior PPA. Because of this, $\mathcal{L}_{\text{rank}}$ is a pairwise ranking loss computed over placement pairs from the same circuit:

$$\mathcal{L}_{\text{rank}} = \sum_{(i,j)} \max\Big(0,\ -(\mathbf{y}_i - \mathbf{y}_j) \cdot \big(f_\theta(\mathbf{x}_i) - f_\theta(\mathbf{x}_j)\big)\Big), \tag{8}$$

where the sum is over all placement pairs (i, j) from the same circuit. Each of the four PPA metrics is oriented so that lower values are better, then normalized to a z-score within each circuit. The hinge incurs zero loss for correctly ranked pairs and a linear penalty for misranked ones, directly optimizing placement ordering. Power varies by less than 2% across configurations of the same circuit, so timing dominates the ranking signal. With $M = 500$ samples per circuit, each epoch compares all placement pairs within each circuit. The comparisons are processed in circuit-specific mini-batches of 32 placements, with each batch compared against cached predictions for all M placements. The MSE term stabilizes learning with an absolute signal, while the ranking loss improves the placement ordering used for candidate selection.

5.4 Differentiable Feature Extraction

The feature extraction layer maps cell positions to the spatial grid and node features in Section 5.1 through differentiable operations. This ensures that gradients $\partial f_\theta / \partial \mathbf{p}$ flow end-to-end from predicted PPA back to cell coordinates. The same differentiable features are used during both training and gradient-guided placement.

Spatial channels. Each of the five channels is computed as a continuous function of cell positions. Cell density (channel 0) uses clamp-based overlap area. This is piecewise linear and differentiable almost everywhere, matching the functional form DREAMPlace uses for its density penalty. Pin density (channel 1) uses Gaussian splatting ($\sigma = 1.5$ bin widths), distributing each pin's contribution smoothly across neighboring bins. Macro occupancy (channel 2) uses sigmoid soft masks ($\sigma = 20$), producing a near-binary mask with a smooth transition at macro boundaries. RUDY-style congestion (channel 3) computes net bounding boxes via the log-sum-exp smooth approximation to max and min (temperature $\gamma = 10$), matching DREAMPlace's wirelength smoothing. Net bounding-box density (channel 4) uses the same Gaussian splatting as channel 1.

Node features. Positions x_i, y_i enter the feature vector (Eq. 5) directly. Average net span s_i uses the same log-sum-exp HPWL approximation as the spatial RUDY channel. Static features (width, height, pin count, net degree) carry zero gradient and require no modification. Channel normalization uses $\mathbf{g}_c / (\max(\mathbf{g}_c) + \epsilon)$, where max is differentiable via PyTorch's `amax`.

The full pipeline from cell positions $\mathbf{p}$ through feature extraction, GAT, CNN, and MLP to predicted PPA is end-to-end differentiable by construction. The gradient $\nabla_{\mathbf{p}} f_\theta$ is available via a single backward pass.

5.5 Gradient-Guided Placement

Given the differentiable surrogate f_θ and a placement $\mathbf{p} = (x_1, y_1, \ldots, x_N, y_N)$, the gradient $\nabla_{\mathbf{p}} f_\theta$ indicates how each cell's position affects the predicted post-route timing objective. PPAPlace exploits this signal in two complementary modes.

Co-objective placement (PPAPlace-CoOpt). The surrogate is injected directly into DREAMPlace's analytical placement loop as a third objective alongside wirelength and density (Algorithm 1). DREAMPlace first runs W warmup iterations to reach a rough solution within the surrogate's training distribution. The total objective then becomes:

$$\mathcal{L}_{\text{total}} = \mathcal{L}_{\text{WL}} + \lambda_d \, \mathcal{L}_{\text{density}} + \lambda_p \, (\widehat{\text{WNS}} + \widehat{\text{TNS}}), \tag{9}$$

where $\widehat{\text{WNS}}$ and $\widehat{\text{TNS}}$ are the timing outputs of $f_\theta(\mathbf{p})$ (power and area are not optimized: area is fixed by the floorplan and power varies by less than 2% across configurations), and λ_p increases linearly from 0 to its target value over the remaining iterations. The gradient reaches macros through both streams and standard cells through the spatial grid's pin-density, RUDY, and net-bounding-box channels, so CoOpt reshapes standard-cell clustering as well as macro placement, which no analytical proxy achieves.

Algorithm 1 PPAPlace-CoOpt: Co-Objective Placement

Require: Predictor f_θ, warmup W, target weight λ_p^*, total iterations N
Initialize positions $\mathbf{p}_0$ randomly
for $t = 1, \ldots, N$ **do**
  $\mathcal{L} \leftarrow \mathcal{L}_{\text{WL}} + \lambda_d \, \mathcal{L}_{\text{density}}$ {standard DREAMPlace}
  if $t > W$ **then**
    $\lambda_p \leftarrow \lambda_p^* \cdot (t - W)/(N - W)$ {linear ramp}
    $\mathcal{L} \leftarrow \mathcal{L} + \lambda_p \, (\widehat{\text{WNS}} + \widehat{\text{TNS}})$ {timing co-objective}
  end if
  $\mathbf{p}_t \leftarrow \mathbf{p}_{t-1} - \eta \, \nabla_{\mathbf{p}} \mathcal{L}$ {GPU-accelerated update}
end for
return placement $\mathbf{p}_N$

Post-placement refinement (PPAPlace-Refine). Starting from any converged placement $\mathbf{p}_0$, projected gradient descent minimizes the surrogate timing objective (Algorithm 2):

$$\mathbf{p}_{t+1} = \Pi_C\Big(\mathbf{p}_t - \alpha \, \nabla_{\mathbf{p}} \big[\widehat{\text{WNS}} + \widehat{\text{TNS}}\big]\Big), \tag{10}$$

where α is the learning rate and Π_C clips to the die bounding box. The refined macros are then legalized to resolve overlaps and enforce boundary constraints, and standard cells are re-optimized with macros fixed. Only the refined macro locations survive this handoff, so Refine is effectively a macro-position method, whereas CoOpt reshapes the full mixed-size placement.

The two modes serve different roles. CoOpt reshapes the global cell topology during placement but requires integration with a specific analytical placer. Refine adjusts positions locally after placement but requires only a legal placement, not access to the source

placer's internals. It applies to any method that produces a placement, including commercial tools whose internals are inaccessible (Section 6.2).

Algorithm 2 PPAPlace-Refine: Post-Placement Refinement

Require: Converged placement $\mathbf{p}_0$, predictor f_θ, learning rate α, steps T
for $t = 1, \ldots, T$ **do**
  Compute differentiable features from $\mathbf{p}_{t-1}$
  $\hat{\mathbf{y}} \leftarrow f_\theta(\text{features})$ {surrogate prediction (<0.1 s)}
  Save $\mathbf{p}_{t-1}$ as $\mathbf{p}^*$ if $\hat{\mathbf{y}}$ has the lowest timing loss so far
  $\mathbf{g} \leftarrow \nabla_{\mathbf{p}}(\widehat{\text{WNS}} + \widehat{\text{TNS}})$ {backprop}
  $\mathbf{p}_t \leftarrow \Pi_C(\mathbf{p}_{t-1} - \alpha\,\mathbf{g})$ {projected gradient step}
end for
Legalize $\mathbf{p}^*$ and re-optimize standard cells
return refined placement $\mathbf{p}^*$

6 Experiments and Results

6.1 Setup

6.1.1 Benchmarks. Experiments use ChiPBench [26] with the Nangate45 library. The ten training circuits are bp_fe, bp_be12, isa_npu, bp_multi, or1200, swerv_wrapper43, vga_lcd, ethernet, dft68, and mor1kx. The five held-out circuits comprise three in-family designs (swerv_wrapper, black_parrot, and bp_be) and two out-of-family designs (ariane133 and ariane136), with no Ariane circuit in training. They match LaMPlace's [6] ChiPBench test set and use the same Hier-RTLMP normalization. The full OpenROAD post-route flow reports WNS (ps), TNS (ns), power (mW), and area (μm^2).

6.1.2 Training. Per training circuit, 1,000 DREAMPlace configurations are sampled by randomizing 10 hyperparameters: target density $\in [0.70, 0.90]$, density weight $\in [10^{-5}, 10^{-3}]$ (log-uniform), learning rate $\in [10^{-3}, 0.032]$ (log-uniform), gamma $\in [2, 10]$, stop overflow $\in \{0.05, 0.07, 0.10, 0.15\}$, wirelength model $\in$ {weighted-average, log-sum-exp}, global-placement iterations $\in \{800, 1000, 1200, 1500\}$, macro halo $\in [0, 10]$ sites, noise ratio $\in [0.01, 0.05]$, and random seed $\in [1, 10^5]$. About 65% converge in DREAMPlace and complete the OpenROAD flow successfully. We retain the first 500 successful placements per circuit through post-GRT, yielding 5,000 training pairs.

The surrogate uses Eq. 7 with $\lambda_r = 0.5$. Mini-batches contain 32 placements grouped by circuit so ranking pairs share a netlist. Adam trains for 200 epochs with a learning rate 5×10^{-4} and weight decay 10^{-5}. Results report mean ± std over three independently seeded runs.

The label fidelity study (Section 4) uses RTLMP placements, while training uses DREAMPlace placements. This fidelity analysis concerns the relationship between flow stages and PPA metrics. Since it uses RTLMP placements, it does not establish placer-independent predictor accuracy. Section 6.4 separately measures the DREAMPlace-trained predictor on RTLMP placements. Repeating the stage study on DREAMPlace placements remains future work.

6.1.3 Baselines and methods. Hier-RTLMP [11] is the ChiPBench reference, normalized to 1.00. DREAMPlace [16] uses ChiPBench's default wirelength-and-density configuration. DREAMPlace 4.0 [15] adds OpenTimer pre-route STA weighting, and we run its open-source release with default parameters. AutoDMP [1] tunes DREAMPlace through multi-objective Bayesian optimization. MaskRegulate [27] uses reinforcement learning with a regularity reward. LaMPlace [6] trains on pre-route STA labels and applies its learned L-mask in the WireMask-EA search framework. Re^{2}MaP [20] uses recursive mixed-size prototyping with hand-engineered costs. Table 2 gives result provenance and flow differences.

All PPAPlace configurations use the same surrogate. Refine (Algorithm 2) applies T=30 projected-gradient steps (α=0.001) on predicted WNS+TNS to the default DREAMPlace placement, followed by legalization and standard-cell re-optimization. CoOpt (Algorithm 1) adds the surrogate after W=200 warmup iterations and linearly ramps λ_p to 0.01. CoOpt+Refine applies Refine after CoOpt.

6.1.4 Hardware and offline cost. Experiments use one NVIDIA RTX A6000 GPU (48 GB), an Intel Xeon Gold 5218R CPU, and 64 GB RAM. DREAMPlace takes ~24 s per GPU configuration. Sixteen concurrent CPU OpenROAD processes produce post-GRT labels in 0.2 h per sample on average, ranging from ~0.05 h to ~0.3 h (Table 1). Pipelined placement and evaluation of all 5,000 labels takes ~63 h of elapsed time, and model training takes ~45 min. This offline cost is amortized: the same dataset and model serve all downstream experiments without additional labeling.

6.2 Main Results

Table 2 reports per-circuit WNS and TNS, the metrics most sensitive to placement quality. Per-design and average congestion are also reported, while power ratios are given in the text below. All these metrics are normalized by Hier-RTLMP's results for more straightforward comparisons (lower is better). Area is fixed by the floorplan and omitted.

Among prior methods, DREAMPlace achieves the lowest HPWL but the worst post-route timing (average WNS 1.61×), confirming the HPWL-to-PPA disconnection reported in ChiPBench [26] and illustrated for swerv_wrapper in Figure 2. Pre-route STA weighting partially bridges this gap: DREAMPlace 4.0 reaches 1.17× WNS / 1.29× TNS and LaMPlace 1.32× / 1.36×, with uneven per-circuit gains (ariane133 WNS drops from 2.90 to 1.65, while near-baseline circuits see marginal changes). AutoDMP (1.20× WNS) and MaskRegulate (1.01× WNS) improve timing through configuration tuning and RL, though MaskRegulate exhibits high variance (ariane133 0.60× vs. ariane136 TNS 3.42×). Re^{2}MaP (0.89× WNS, 0.61× TNS) is the strongest prior method, with particularly notable TNS on black_parrot (0.02×) and bp_be (0.51×).

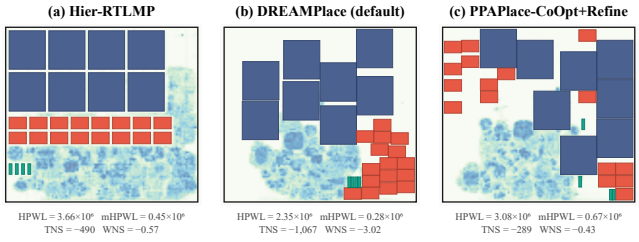


Figure 2: Placement comparison on `swerv_wrapper`: (a) Hier-RTLMP, (b) DREAMPlace, (c) PPAPlace-CoOpt+Refine.

Table 2: Post-route timing and global-routing wire-density congestion, each normalized per circuit by Hier-RTLMP (lower is better). PPAPlace rows report mean ± std over 3 independently seeded runs; baselines are deterministic or from published results. Bold : best; underline : second best. †: from [6]; ‡: from ChiPBench [26]; §: ratios computed from [20] (same OpenROAD flow; std-cell placement differs).

	swerv_wrap			ariane133			black_parrot			bp_be			ariane136			**Average**		
Method	WNS	TNS	Cong	WNS	TNS	Cong	WNS	TNS	Cong	WNS	TNS	Cong	WNS	TNS	Cong	WNS	TNS	Cong
Prior placement methods																		
Hier-RTLMP [11]	1.00	1.00	1.00	1.00	1.00	1.00	1.00	1.00	**1.00**	1.00	1.00	1.00	1.00	1.00	1.00	1.00	1.00	1.00
DREAMPlace‡ [16]	1.13	0.93	1.04	2.90	0.61	0.98	0.94	1.82	1.05	0.87	0.86	1.06	2.23	5.37	**0.94**	1.61	1.92	1.01
DREAMPlace 4.0 [15]	1.02	0.88	1.05	1.65	0.58	0.99	0.91	1.30	1.06	0.82	0.83	1.08	1.45	2.85	0.96	1.17	1.29	1.03
AutoDMP‡ [1]	1.43	1.47	1.09	1.44	1.76	**0.91**	1.01	0.85	1.06	**0.49**	1.03	1.18	1.62	3.69	**0.94**	1.20	1.76	1.04
MaskRegulate‡ [27]	1.02	0.84	1.00	**0.60**	**0.24**	1.02	0.91	0.14	1.01	0.85	0.83	**0.88**	1.67	3.42	1.04	1.01	1.09	0.99
LaMPlace† [6]	1.55	1.21	0.99	1.48	1.77	1.10	1.18	1.28	1.02	1.25	1.38	1.05	1.12	1.15	0.97	1.32	1.36	1.03
Re²MaP§ [20]	0.78	0.62	**0.88**	0.93	0.99	1.02	0.98	**0.02**	1.01	0.85	0.51	0.90	0.92	0.91	1.00	0.89	0.61	**0.96**
Gradient-guided placement (ours)																		
PPAPlace-Refine	1.02±.03	0.85±.03	1.03±.02	2.15±.08	0.52±.04	0.98±.02	0.88±.02	1.20±.05	1.04±.02	0.78±.02	0.72±.04	1.05±.03	1.65±.06	3.50±.10	0.95±.02	1.30±.04	1.36±.05	1.01±.02
PPAPlace-CoOpt	0.82±.03	0.68±.04	1.01±.02	0.90±.04	0.50±.03	0.96±.02	0.84±.02	0.22±.04	1.03±.02	0.72±.02	0.55±.04	1.05±.03	0.88±.03	0.90±.05	**0.94±.02**	0.83±.03	0.57±.04	1.00±.02
PPAPlace-CoOpt+Refine	**0.76±.03**	**0.59±.04**	1.00±.02	0.85±.04	0.42±.03	0.96±.02	**0.81±.02**	0.15±.04	1.03±.02	0.65±.02	**0.48±.05**	1.04±.03	**0.84±.03**	**0.82±.04**	**0.94±.02**	**0.78±.03**	**0.49±.04**	0.99±.02

PPAPlace-Refine alone applies gradient descent to DREAMPlace's default placement, reducing average WNS from 1.61 to 1.30 and TNS from 1.92 to 1.36. This improves over untuned DREAMPlace but the local nature of gradient refinement limits gains on circuits where the starting topology is already misaligned (ariane133 WNS remains 2.15×).

PPAPlace-CoOpt injects the surrogate co-objective into DREAMPlace's loop, achieving average WNS of 0.83 and TNS of 0.57. The gap over DREAMPlace 4.0 demonstrates the value of post-route supervision over pre-route STA. Post-DRT reports confirm zero design-rule-check violations and routed wirelength within 3% of default, so the co-objective does not degrade routability.

CoOpt+Refine's WNS/TNS ratios (improvements) are 0.74/0.41 (26%/59%) for in-family circuits, 0.85/0.62 (15%/38%) for out-of-family Ariane circuits, and 0.78/0.49 (22%/51%) overall. This surpasses Re²MaP's algorithmic approach (0.89/0.61) on both metrics, demonstrating that a learned post-route objective can outperform hand-engineered placement heuristics. Unlike MaskRegulate's circuit-specific gains, CoOpt+Refine improves consistently across all test circuits (Figure 3(a)). It achieves the best per-circuit result on 6 of 10 circuit–metric pairs and the best average on both WNS and TNS. On the remaining four pairs, it yields the second-best result except for the TNS of black_parrot. The power ratios (0.99–1.02×) confirm that PPAPlace's timing improvements do not increase power. On the same hardware, standard DREAMPlace takes about 24 seconds per placement, while CoOpt completes in under one minute, giving a placement-time overhead below 2.5×.

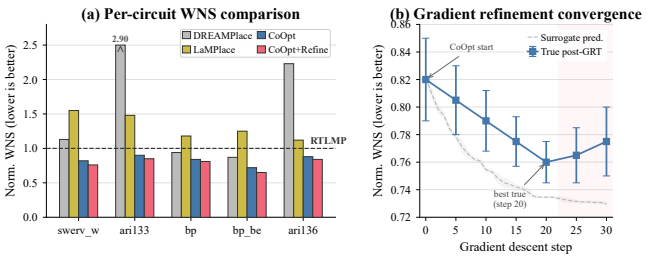


Figure 3: (a) Per-circuit normalized WNS (Table 2). (b) True vs. surrogate WNS during gradient refinement on swerv_wrapper.

All the results reported in Table 2 are final post-DRT PPA metrics. Although the surrogate is trained on post-GRT labels, the improvements carry through to full detailed routing. This is consistent with the fidelity analysis in Section 4, where post-GRT rankings preserve post-DRT outcomes with $\bar{\rho} = 0.86$.

6.3 Gradient Quality

Gradient-guided placement requires directionally accurate gradients. `torch.autograd.gradcheck` confirms numerical correctness (relative error $< 10^{-6}$) on 30 test placements. To assess directional alignment, we perturb converged placements along 100 random directions per circuit and measure true PPA changes via post-GRT. Table 3 reports cosine similarity between surrogate and true gradients: average 0.53 for WNS and 0.46 for TNS, positive on all circuits.

Table 3: Cosine similarity between surrogate gradient and true post-GRT PPA change (100 perturbation directions per circuit).

	swerv	ari133	black	bp_be	ari136	Avg.
cos(∇ WNS)	0.49	0.58	0.51	0.62	0.43	0.53
cos(∇ TNS)	0.43	0.49	0.46	0.53	0.37	0.46

Figure 3(b) validates this on swerv_wrapper: gradient descent reduces true post-GRT WNS from 0.82 to 0.76 over 20 steps, closely tracked by the surrogate. Beyond step 22, true WNS rises as the placement exits the training distribution. The refinement loop returns the checkpoint with the lowest surrogate loss across all T steps, mitigating moderate overshoot.

We sweep $\lambda_p \in \{0.001, 0.005, 0.01, 0.05\}$ on swerv_wrapper, yielding WNS of $\{0.92, 0.87, 0.82, 0.90\}$; $\lambda_p = 0.01$ provides the best trade-off and is reused unchanged across all test circuits. For refinement steps, $T \in \{10, 20, 30, 50\}$ gives WNS $\{0.79, 0.77, 0.76, 0.77\}$, with T=30 best and T=50 slightly worse. The learning rate α=0.001 is selected by grid search.

6.4 Generalization

Table 4 evaluates 500 held-out DREAMPlace placements per circuit. Average WNS Spearman ρ is 0.77, Kendall τ is 0.58, and top-5 accuracy is 68% versus 1% for random selection. Accuracy ranges from $\rho = 0.72$ on ariane136 to 0.83 on bp_be. Three test circuits

share training-circuit lineages (bp_be/bp_be12, swerv_wrapper/swerv_wrapper43, black_parrot/bp_fe). Shared design lineages can make these three cases easier than unseen design families. The Ariane circuits, both absent from training, reach $\rho = 0.81$ and 0.72, respectively, with ariane136 lowest.

Table 4: Surrogate generalization: ranking accuracy on held-out placements and cross-placer transfer to RTLMP.

Metric	swerv	ari133	black	bp_be	ari136	bp_fe[t]	ether[t]	Avg.
Held-out (DREAMPlace)								
ρ	0.74	0.81	0.77	0.83	0.72	N/A	N/A	0.77
τ	0.56	0.62	0.55	0.65	0.51	N/A	N/A	0.58
Top-5	60%	80%	60%	80%	60%	N/A	N/A	68%
Cross-placer (RTLMP)								
ρ_{WNS}	0.63	0.58	0.60	0.72	0.54	0.68	0.55	0.61
ρ_{TNS}	0.57	0.54	0.53	0.66	0.50	0.60	0.52	0.56

[t]Training circuit (cross-placer only).

Leave-one-circuit-out (LOCO) cross-validation (Figure 4(a)) trains on 9 circuits and evaluates on the tenth. Per-circuit τ ranges from 0.07 on isa_npu to 0.28 on mor1kx, which is sufficient to identify above-average placements.

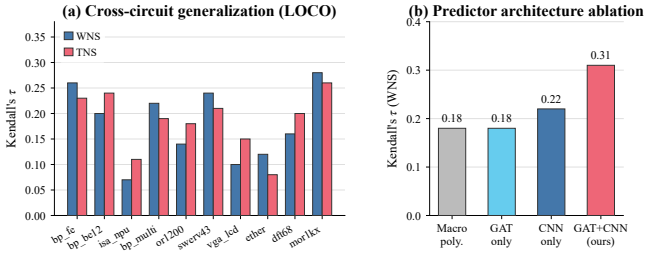


Figure 4: (a) Leave-one-circuit-out (LOCO) cross-circuit generalization (Kendall's τ). (b) Predictor architecture ablation (Kendall's τ).

The cross-placer rows of Table 4 apply the DREAMPlace-trained predictor to unseen RTLMP placements. WNS ρ is 0.61 versus 0.77 on DREAMPlace and remains significant ($p < 0.01$) on every circuit. This drop indicates a placement-distribution gap rather than placer independence.

Table 5 evaluates the same checkpoint zero-shot on 100 IBM 45 nm Superblue16/18 placements, 50 per circuit. WNS/TNS Spearman ρ is 0.68/0.62, with 56% top-5 accuracy. Against LaMPlace [6], PPAPlace ranks second in WNS, first on superblue16 TNS, and within 3.2% on superblue18 TNS.

6.5 Ablation Studies

6.5.1 Label Fidelity and Representation. The upper section of Table 6 crosses two label stages with two representations under identical training. LaMPlace's macro-only + pre-route STA setting has weak ranking agreement ($\tau = 0.08$). Both axes contribute independently. Post-GRT doubles macro-only τ from 0.08 to 0.18, GAT+CNN raises pre-route τ from 0.08 to 0.13, and combining both reaches 0.31.

Table 5: Final timing on Superblue16/18, with baselines from LaMPlace [6]. WNS is in 10^3 ps and TNS in 10^5 ps. Values closer to zero are better.

	superblue16		superblue18	
Method	WNS	TNS	WNS	TNS
DREAMPlace [16]	−107.05	−1526.10	−88.11	−751.27
WireMask-EA [22]	−635.89	−18343.30	−78.25	−406.01
ChiPFormer [12]	−322.05	−15426.07	−80.57	−378.90
LaMPlace [6]	−36.87	−1514.73	−66.93	−426.91
PPAPlace zero-shot	−68.50	−1380.50	−73.40	−440.50

Table 6: Ablation on labels and representation (Kendall's τ, WNS). Upper: two label stages × two representations. Lower: label stages with GAT+CNN fixed.

Training labels	Architecture	τ (WNS)	Top-1
Label stage × representation			
Pre-route STA	Macro-only poly.	0.08±.02	n/a
Pre-route STA	GAT+CNN	0.13±.03	26%
Post-GRT	Macro-only poly.	0.18±.02	n/a
Post-GRT	GAT+CNN	**0.31±.02**	**52%**
Label stage (GAT+CNN fixed)			
Post-CTS	GAT+CNN	0.16±.03	30%
Post-DRT	GAT+CNN	0.34±.02	56%

With GAT+CNN fixed, τ rises from 0.13 with pre-route STA to 0.16 with post-CTS and 0.31 with post-GRT, isolating the supervision stage. Post-DRT reaches 0.34 but costs 3.7 hours per label versus 0.20 hours for post-GRT, which supports more training data under the same offline budget.

6.5.2 Predictor Architecture. Figure 4(b) compares four post-GRT variants: macro-only polynomial ($\tau = 0.18$), CNN-only (0.22), GAT-only (0.18), and GAT+CNN (0.31). The combined model outperforms either stream alone, showing that density patterns and netlist connectivity provide complementary information.

7 Conclusion

PPAPlace demonstrates that a differentiable surrogate trained on post-GRT labels can provide gradient-based PPA feedback that no analytical proxy achieves. CoOpt+Refine improves average WNS and TNS by 22% and 51% over Hier-RTLMP on five held-out circuits.

Despite the performance gains, several limitations suggest promising future directions. Training uses Nangate45, while the zero-shot Superblue test uses the IBM 45 nm library. Broader validation across technology nodes and standard-cell libraries remains a natural next step. The raw refinement trajectory exhibits out-of-distribution degradation after about 20 steps (Figure 3(b)). Distribution-aware stopping or trust-region constraints could improve robustness. Cross-placer transfer ($\rho = 0.61$) lags behind within-placer accuracy (0.77). Targeted fine-tuning or domain-adaptation techniques could narrow this gap. Finally, CoOpt currently requires integration with a differentiable placer. Extending the co-objective paradigm to commercial tools that expose only final placements remains an open challenge that the input-compatible Refine mode partially addresses.

Acknowledgments

Supported by the Natural Sciences and Engineering Research Council of Canada (NSERC) (RES0048688, RES0051374, and RES0054326) and Alberta Innovates (RES0053965).

References

[1] Anthony Agnesina, Puranjay Rajvanshi, Tian Yang, Geraldo Pradipta, Austin Jiao, Ben Keller, Brucek Khailany, and Haoxing Ren. 2023. AutoDMP: Automated DREAMPlace-based Macro Placement. In *Proceedings of the International Symposium on Physical Design (ISPD)*. ACM, 149–157. doi:10.1145/3569052.3578923

[2] Tutu Ajayi, Vidya A. Chhabria, Mateus Fogaça, Soheil Hashemi, Abdelrahman Hosny, Andrew B. Kahng, Minsoo Kim, Jeongsup Lee, Uday Mallappa, Marina Neseem, Geraldo Pradipta, Sherief Reda, Mehdi Saligane, Sachin S. Sapatnekar, Carl Sechen, Mohamed Shalan, William Swartz, Lutong Wang, Zhehong Wang, Mingyu Woo, and Bangqi Xu. 2019. INVITED: Toward an Open-Source Digital Flow: First Learnings from the OpenROAD Project. In *Proceedings of the 56th Annual Design Automation Conference (DAC)*. ACM, 1–4. doi:10.1145/3316781.3326334

[3] Cadence. 2021. Cadence Cerebrus Intelligent Chip Explorer. https://www.cadence.com/en_US/home/tools/digital-design-and-signoff/soc-implementation-and-floorplanning/cerebrus-intelligent-chip-explorer.html. Accessed August 12, 2026.

[4] Yifan Chen, Jing Mai, Xiaohan Gao, Muhan Zhang, and Yibo Lin. 2023. MacroRank: Ranking Macro Placement Solutions Leveraging Translation Equivariancy. In *Asia and South Pacific Design Automation Conference (ASP-DAC)*. ACM, 258–263. doi:10.1145/3566097.3567899

[5] Chung-Kuan Cheng, Andrew B. Kahng, Ilgweon Kang, and Lutong Wang. 2019. RePlAce: Advancing Solution Quality and Routability Validation in Global Placement. *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems* 38, 9 (2019), 1717–1730. doi:10.1109/TCAD.2018.2859220

[6] Zijie Geng, Jie Wang, Ziyan Liu, Siyuan Xu, Zhentao Tang, Shixiong Kai, Mingxuan Yuan, Jianye Hao, and Feng Wu. 2025. LaMPlace: Learning to Optimize Cross-Stage Metrics in Macro Placement. In *International Conference on Learning Representations (ICLR)*. https://proceedings.iclr.cc/paper_files/paper/2025/hash/04c0399a47ee4107cd03b08f1f8c3eeb-Abstract-Conference.html

[7] Zijie Geng, Jie Wang, Ziyan Liu, Siyuan Xu, Zhentao Tang, Mingxuan Yuan, Jianye Hao, Yongdong 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) (Proceedings of Machine Learning Research, Vol. 235)*. PMLR, 15402–15417. https://proceedings.mlr.press/v235/geng24b.html

[8] Yunbo Hou, Haoran Ye, Yingxue Zhang, Siyuan Xu, and Guojie Song. 2024. 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)*. ACM, 1085–1095. doi:10.1145/3637528.3671895

[9] Meng-Kai Hsu, Valeriy Balabanov, and Yao-Wen Chang. 2013. TSV-Aware Analytical Placement for 3-D IC Designs Based on a Novel Weighted-Average Wirelength Model. *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems* 32, 4 (2013), 497–509. doi:10.1109/TCAD.2012.2226584

[10] Andrew B. Kahng, Jens Lienig, Igor L. Markov, and Jin Hu. 2022. *VLSI Physical Design: From Graph Partitioning to Timing Closure* (2nd ed.). Springer. doi:10.1007/978-3-030-96415-3

[11] Andrew B. Kahng, Ravi Varadarajan, and Zhiang Wang. 2024. Hier-RTLMP: A Hierarchical Automatic Macro Placer for Large-Scale Complex IP Blocks. *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems* 43, 5 (2024), 1552–1565. doi:10.1109/TCAD.2023.3346284

[12] 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) (Proceedings of Machine Learning Research, Vol. 202)*. PMLR, 18346–18364. https://proceedings.mlr.press/v202/lai23c.html

[13] Yao Lai, Yao Mu, and Ping Luo. 2022. MaskPlace: Fast Chip Placement via Reinforced Visual Representation Learning. In *Advances in Neural Information Processing Systems (NeurIPS)*, Vol. 35. 24019–24030. doi:10.52202/068431-1744

[14] 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) (Proceedings of Machine Learning Research, Vol. 267)*. PMLR, 33499–33514. https://proceedings.mlr.press/v267/lee25y.html

[15] Peiyu Liao, Siting Liu, Zhitang Chen, Wenlong Lv, Yibo Lin, and Bei Yu. 2022. DREAMPlace 4.0: Timing-Driven Global Placement with Momentum-Based Net Weighting. In *Design, Automation and Test in Europe Conference (DATE)*. IEEE, 939–944. doi:10.23919/DATE54114.2022.9774725

[16] 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)*. ACM, 1–6. doi:10.1145/3316781.3317803

[17] Jingwei Lu, Pengwen Chen, Chin-Chih Chang, Lu Sha, Dennis Jen-Hsin Huang, Chin-Chi Teng, and Chung-Kuan Cheng. 2014. ePlace: Electrostatics Based Placement Using Nesterov's Method. In *Proceedings of the 51st Annual Design Automation Conference (DAC)*. ACM, 1–6. doi:10.1145/2593069.2593133

[18] Igor L. Markov. 2024. Reevaluating Google's Reinforcement Learning for IC Macro Placement. *Commun. ACM* 67, 11 (2024), 60–71. doi:10.1145/3676845

[19] Azalia Mirhoseini, Anna Goldie, Mustafa Yazgan, Joe Wenjie Jiang, Ebrahim Songhori, Shen Wang, Young-Joon Lee, Eric Johnson, Omkar Pathak, Azade Nova, 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

[20] Yunqi Shi, Xi Lin, Zhiang Wang, Siyuan Xu, Shixiong Kai, Yao Lai, Chengrui Gao, Ke Xue, Mingxuan Yuan, Chao Qian, and Zhi-Hua Zhou. 2026. Re^{2}MaP: Macro Placement by Recursively Prototyping and Packing Tree-based Relocating. *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems* (2026). Early Access. doi:10.1109/TCAD.2026.3699957

[21] Yunqi Shi, Siyuan Xu, Shixiong Kai, Xi Lin, Ke Xue, Mingxuan Yuan, and Chao Qian. 2025. Timing-Driven Global Placement by Efficient Critical Path Extraction. In *Design, Automation and Test in Europe Conference (DATE)*. IEEE, 1–7. doi:10.23919/DATE64628.2025.10993273

[22] Yunqi Shi, Ke Xue, Lei Song, and Chao Qian. 2023. Macro Placement by Wire-Mask-Guided Black-Box Optimization. In *Advances in Neural Information Processing Systems (NeurIPS)*, Vol. 36. 6825–6843. doi:10.52202/075280-0299

[23] Peter Spindler and Frank M. Johannes. 2007. Fast and Accurate Routing Demand Estimation for Efficient Routability-Driven Placement. In *Design, Automation and Test in Europe Conference (DATE)*. IEEE, 1–6. doi:10.1109/DATE.2007.364463

[24] Synopsys. 2020. DSO.ai: AI-Driven Design Applications. https://www.synopsys.com/ai/ai-powered-eda/dso-ai.html. Accessed August 12, 2026.

[25] Petar Veličković, Guillem Cucurull, Arantxa Casanova, Adriana Romero, Pietro Liò, and Yoshua Bengio. 2018. Graph Attention Networks. In *International Conference on Learning Representations (ICLR)*. https://openreview.net/forum?id=rJXMpikCZ

[26] Zhihai Wang, Zijie Geng, Zhaojie Tu, Jie Wang, Yuxi Qian, Zhexuan Xu, Ziyan Liu, Siyuan Xu, Zhentao Tang, Shixiong Kai, Mingxuan Yuan, Jianye Hao, Bin Li, and Feng Wu. 2025. Benchmarking End-To-End Performance of AI-Based Chip Placement Algorithms. In *Advances in Neural Information Processing Systems (NeurIPS), Datasets and Benchmarks Track*, Vol. 38. 22670–22696. doi:10.52202/085713-0673

[27] Ke Xue, Ruo-Tong Chen, Xi Lin, Yunqi Shi, Shixiong Kai, Siyuan Xu, and Chao Qian. 2024. Reinforcement Learning Policy as Macro Regulator Rather than Macro Placer. In *Advances in Neural Information Processing Systems (NeurIPS)*, Vol. 37. 140565–140588. doi:10.52202/079017-4462

[28] Ke Xue, Ruo-Tong Chen, Rong-Xi Tan, Xi Lin, Yunqi Shi, Siyuan Xu, Mingxuan Yuan, and Chao Qian. 2026. BBOPlace-Bench: Benchmarking Black-Box Optimization for Chip Placement. *IEEE Transactions on Evolutionary Computation* (2026). Early Access. doi:10.1109/TEVC.2026.3712410

[29] Ruizhe Zhong, Junjie Ye, Zhentao Tang, Shixiong Kai, Mingxuan Yuan, Jianye Hao, and Junchi Yan. 2024. PreRoutGNN for Timing Prediction with Order Preserving Partition: Global Circuit Pre-Training, Local Delay Learning and Attentional Cell Modeling. *Proceedings of the AAAI Conference on Artificial Intelligence* 38, 15 (2024), 17087–17095. doi:10.1609/aaai.v38i15.29653