HyPAS : A Hybrid Optimization Framework for Placement and Sizing Co-Optimization

Fangzhou Liu[1,†], Wuqian Tang[2,†], Bo-Ying Wang[2], An-Chieh Shen[2], Han-Wen Tsao[2], Yuyang Ye[1], Yun Shao[3], Chun-Yao Wang[2], Bei Yu[1]
[1]Chinese University of Hong Kong [2]National Tsing Hua University [3]Giga Design Automation

Abstract

Incremental placement optimization improves PPA by refining cell locations, gate sizing, and buffering under physical constraints. Conventional incremental flows, composed of discrete heuristic stages with separate timing analyses, often converge slowly and yield inconsistent PPA gains due to the lack of a unified formulation. Recent differentiable placement methods enable scalable gradient-based refinement on smooth surrogates of wirelength, density, and timing but remain fragile near legality and timing-closure boundaries. To combine their strengths, we propose HyPAS, a hybrid framework that integrates discrete sign-off optimization with differentiable gradient refinement for placement and sizing co-optimization. The discrete stage, implemented with OpenROAD, ensures legality, timing closure, and buffer-aware repair; while the differentiable stage, built on DREAMPlace, performs gradient-guided placement and sizing using a GNN-based timing-power surrogate. A straight-through estimator (STE) bridges continuous gradients with discrete library parameters, enabling end-to-end physical co-optimization. Experiments show HyPAS delivers superior PPA gains with competitive runtime against 2025 ICCAD CAD Contest results and SOTA methods.

1 Introduction

Incremental placement optimization is a crucial stage in modern physical design. After global placement converges, designers refine cell placement to improve timing, power, and wirelength while ensuring legality and functional correctness. This step is essential for sign-off quality as it directly balances performance, power, and area (PPA) [1, 2]. Unlike global placement focusing on coarse wirelength and density, incremental placement operates in a highly constrained solution space, where most cells are legalized and only small adjustments are allowed [3]. It requires close coordination among cell movement, sizing, and buffering, since any change can immediately impact timing paths, power characteristics, and physical legality [4]. With millions of interacting instances in advanced designs, this optimization is both computationally intensive and tightly coupled across electrical and geometric domains.

In both industry and academia, physical design flows operate on inherently discrete variables: standard cells snap to fixed sites, gate sizes and threshold voltages are chosen from finite libraries, and timing or power is evaluated with non-differentiable models. This discreteness makes the design space vast yet discontinuous, pushing conventional methods to rely on local, rule-based heuristics that iteratively repair neighboring cells through discrete updates. Representative approaches include STA-driven gate sizing and buffering in OpenROAD[5], physically aware buffer insertion from finite candidate sites[6, 7], and Lagrangian-relaxation frameworks for sizing, buffering, and pin swapping [8]. While these methods preserve physical constraints, their heuristic nature imposes three structural limits: the search scope is local, timing/power/wirelength objectives are optimized independently, and the lack of a unified driver hampers global coordination. Consequently, each incremental update incurs costly STA and legalization cycles, and without global feedback, the flow often oscillates between repair and rollback, causing the slow convergence observed in industrial incremental flows [4, 9].

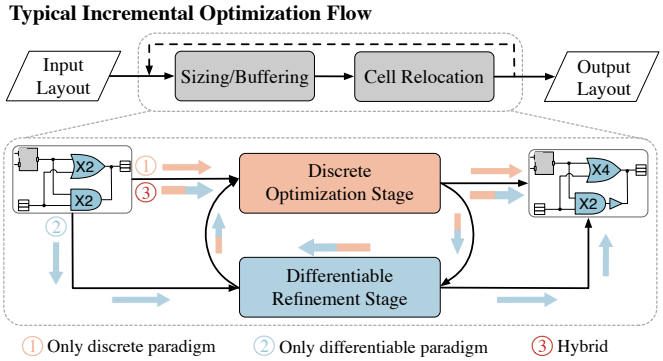


Figure 1: Three different paradigms for incremental placement: ① uses discrete optimizations, ② uses differentiable refinements, ③ hybrid integration in HyPAS .

With the rise of GPU acceleration and machine learning, differentiable optimization has emerged as a powerful approach to relax discrete EDA problems into continuous domains for gradient-based solutions. In placement and sizing, physical effects such as wirelength, density, and delay can be smoothly approximated as differentiable surrogates, enabling gradient-based refinement over otherwise discrete design variables. Frameworks like DREAMPlace [10], XPlace [11], and INSTA [12] exploit analytical gradients and GPU parallelism for large-scale placement, with later works [13–15] extending this to timing-aware objectives. Learning-based approaches [9, 16] further enhance analytical gradients through data-driven surrogates that learn continuous embeddings. However, these methods face practical limitations: abrupt gradient shifts near sign-off boundaries cause oscillatory movements and impede convergence [17], continuous relaxation obscures legality and introduces costly density violations, and sensitivity to initialization and hyperparameters limits robustness across process technologies and design scales.

[†]These authors contributed equally to this work.
*This work is supported by The Research Grants Council of Hong Kong SAR (No. CUHK14211824, CUHK14210723, and T46-415/25-R).

DAC '26, July 26–29, 2026, Long Beach, CA, USA

ACM ISBN 979-8-4007-2254-7/2026/07
https://doi.org/10.1145/3770743.3804137

Building on these insights, we propose HyPAS, a hybrid framework that unifies a discrete stage and a differentiable stage for efficient and robust incremental placement. As depicted in Figure 1, HyPAS integrates a **discrete stage** that enforces physical feasibility through sizing, buffering, and cell relocation, and a **differentiable stage** that performs scalable, gradient-based refinement for global optimization. The discrete stage ensures a physically valid and sign-off-quality foundation, while the differentiable stage refines placement continuously to explore broader design spaces. By combining their complementary strengths, HyPAS bridges heuristic discrete optimization and unstable differentiable methods, achieving globally guided yet physically realizable PPA improvements. In practice, the discrete stage is implemented in OpenROAD for cell relocation and timing repair, and the differentiable stage builds on DREAMPlace with GNN-based timing and power surrogates. The key contributions of this work include:

- We propose HyPAS, a hybrid framework for multi-objective optimization that integrates discrete sign-off procedures with gradient-based refinement.
- We introduce differentiable gate sizing via STE over discrete drive-strength and VT options.
- We develop a GNN-based differentiable timing-power surrogate that guides multi-objective refinement and provides gradient feedback to steer STE-based sizing.
- We evaluate HyPAS on 2025 ICCAD CAD Contest Problem C benchmarks (2K-170K cells), achieving significant PPA and runtime improvements over all top-three contest winners.

2 Preliminaries

2.1 Optimization Paradigms

Incremental placement refines a legalized layout to improve PPA while maintaining design legality. In the discrete paradigm, optimization alternates among placement, sizing, and buffering to balance timing, power, and wirelength. Although rule-based heuristics ensure legality, they explore limited neighborhoods and converge slowly due to repeated STA and legalization. In contrast, the differentiable paradigm relaxes discrete choices into continuous surrogates, enabling GPU-accelerated gradient optimization. It employs smooth wirelength and density formulations, such as log-sum-exp and electrostatic models. Differentiable placers like DREAMPlace [10] minimize:

$$\min_{x,y} \sum_{e} \mathrm{WL}(e) + \lambda\, \mathrm{D}(\boldsymbol{x}, \boldsymbol{y}). \quad (\lambda: \text{ density penalty weight}) \tag{1}$$

Such relaxations can also be applied to gate sizing for timing-power optimization [4], but post-quantization is required to ensure legality.

2.2 Problem Formulation

Given a legal seed placement, our goal is to perform incremental optimization using three operator types: gate sizing, buffer or inverter-pair insertion, and cell relocation. Each cell is defined by its position (x, y) and cell type s. The problem is formulated as

$$\begin{aligned} \min_{\boldsymbol{x},\boldsymbol{y},\boldsymbol{s}} \quad & F_{\mathrm{obj}}(\boldsymbol{x},\boldsymbol{y},\boldsymbol{s}) \\ & = \alpha\, F_{\mathrm{timing}}(\boldsymbol{x},\boldsymbol{y},\boldsymbol{s}) + \beta\, F_{\mathrm{power}}(\boldsymbol{x},\boldsymbol{y},\boldsymbol{s}) + \gamma\, F_{\mathrm{wire}}(\boldsymbol{x},\boldsymbol{y}). \end{aligned} \tag{2}$$

Here, F_{timing} represents the total negative slack (TNS), F_{power} the total power consumption, including both dynamic and leakage components, and F_{wire} the total half-perimeter wirelength (HPWL). The weights (α, β, γ) balance timing, power, and wirelength objectives.

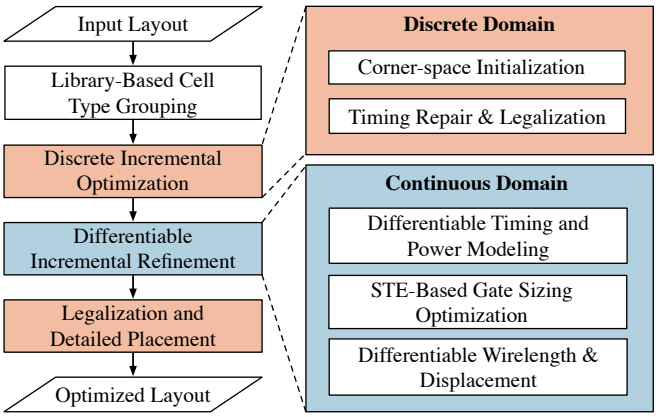


Figure 2: Overall workflow of the HyPAS framework.

Additional penalties for cell displacement and runtime are applied to restrict excessive movement and promote efficient optimization.

3 Algorithms

This section provides a detailed description of the **HyPAS** framework, with its overall workflow illustrated in Figure 2. We first perform library-based cell type grouping to identify permissible sizing candidates from the standard-cell library. Then, a discrete incremental optimization conducts sizing, buffering, and relocation to form a legal, timing-closed foundation. Subsequently, differentiable refinement jointly optimizes placement and sizing, driven by differentiable timing and power models that replace conventional STA and propagate gradients via STE. Finally, discrete legalization and detailed placement are applied to ensure sign-off accuracy.

3.1 Library-Based Cell Type Grouping

To enable safe and legal gate sizing, standard cells (hereafter referred to as cell types) and related parameters are extracted from LIB/LEF files and classified by structural and logical features. Cell types are grouped if they share the same ordered pin set, output function, cell height, and a discrete width-group index g_w. The width grouping g_w clusters cells with similar widths to prevent layout overlaps caused by large width increases during sizing, which often lead to numerous hard-to-fix violations. Each group, denoted as $\mathcal{D}$, allows safe interchange of cell types for drive-strength resizing.

Unlike conventional tools that restrict sizing to the same threshold-voltage (VT) option, our approach jointly considers drive strength and VT. We adopt three VT options: Regular (R), Low (L), and Super-Low (SL), which share the same logic and geometry but differ in delay-leakage trade-offs, enabling joint timing-power optimization [18]. Hence, the candidate set for each cell instance is defined as:

$$\mathcal{S} = \big\{ (d, v) \,\big|\, d \in \mathcal{D},\, v \in \{\mathrm{R}, \mathrm{L}, \mathrm{SL}\} \big\}. \tag{3}$$

Example 1 (Candidate cell types for resizing `AND3x1_R`)**.** Consider four `AND3` cells that have the same pin configuration, logic function, and cell height, but different widths and VT options:

$$\begin{aligned} &\texttt{AND3x1\_R}: \ \text{width} = 0.324 \ \rightarrow \ (g_w = 0), \quad \text{VT} = R; \\ &\texttt{AND3x1\_L}: \ \text{width} = 0.324 \ \rightarrow \ (g_w = 0), \quad \text{VT} = L; \\ &\texttt{AND3x2\_R}: \ \text{width} = 0.378 \ \rightarrow \ (g_w = 0), \quad \text{VT} = R; \\ &\texttt{AND3x4\_R}: \ \text{width} = 0.756 \ \rightarrow \ (g_w = 1), \quad \text{VT} = R. \end{aligned}$$

The `AND3x1` and `AND3x2` cells have similar widths and form $\mathcal{D}^{(0)} = \{\texttt{AND3x1}, \texttt{AND3x2}\}$, while the wider `AND3x4` cell belongs

to another group ($g_w = 1$) and is excluded. An instance using `AND3x1_R` therefore has the candidate set $\mathcal{S}_{\texttt{AND3x1\_R}} = \{(d, v) \mid d \in \mathcal{D}^{(0)}, v \in \{\text{R}, \text{L}, \text{SL}\}\}$, where each drive option d can further pair with different VT options v (e.g., `AND3x1_L`, `AND3x2_SL`).

3.2 Discrete Incremental Optimization

The initial input already provides a legal layout. The discrete optimization is performed to improve PPA through cell relocation, sizing, and buffering, and to produce an optimization-friendly initialization that enables stable refinement in the differentiable stage.

We first apply a rule-based corner-space initialization. Starting from the existing layout, we create three copies by uniformly replacing the VT component of each cell's corresponding cell type with one of {R, L, SL}. In each copy, all cells adopt the same VT type, while original placement and drive-strength are preserved. This provides consistent starting points, avoids cross-corner inconsistencies, and follows the industry practice of starting with RVT cells (lowest leakage) and gradually introducing L/SVT cells for timing improvement. Since both timing and power are critical in multi-objective optimization, we explore VT corners in parallel to broaden the search space. Subsequent stages run concurrently on these layouts, and the best result is selected as the final design.

To improve timing, we use the `repair_timing` command in OpenROAD [5], which applies physically aware, greedy repair via gate sizing [19] and buffer insertion [6, 7]. Topology-altering actions (e.g., pin swapping, gate cloning) and buffer removal are disabled to preserve logical structure and placement legality. This constrained repair reduces timing cost with limited power and wirelength overhead, producing a timing-clean layout for subsequent refinement.

Residual placement deviations may persist after timing repair. To resolve these perturbations, we apply a legalization-oriented detailed placement refinement [20], adjusting cells within legal rows to restore spatial consistency. This step mitigates displacement penalties by snapping each cell to its nearest legal site:

$$(\boldsymbol{x}', \boldsymbol{y}') = \arg \min_{(\boldsymbol{x}', \boldsymbol{y}') \in \mathcal{L}} |(\boldsymbol{x}', \boldsymbol{y}') - (\boldsymbol{x}, \boldsymbol{y})|_2^2, \tag{4}$$

where $\mathcal{L}$ denotes the set of legal placement sites. The refinement yields a stable, well-conditioned layout for subsequent differentiable optimization, though its heuristic nature may still lead to local optima.

3.3 Differentiable Incremental Refinement

Inspired by DREAMPlace [10], we develop a differentiable refinement method for placement and sizing. Starting from the layout produced by the discrete incremental optimization, the objective is:

$$\begin{aligned} F_{\text{obj}}(\boldsymbol{x}, \boldsymbol{y}, \boldsymbol{s}) = \alpha\, \widehat{F}_{\text{timing}}(\boldsymbol{x}, \boldsymbol{y}, \boldsymbol{s}; \theta_t) + \beta\, \widehat{F}_{\text{power}}(\boldsymbol{x}, \boldsymbol{y}, \boldsymbol{s}; \theta_p) \\ + \gamma\, F_{\text{wire}}(\boldsymbol{x}, \boldsymbol{y}) + \lambda_{\text{disp}}\, F_{\text{disp}}(\boldsymbol{x}, \boldsymbol{y}) + \lambda_{\text{dens}}\, F_{\text{dens}}(\boldsymbol{x}, \boldsymbol{y}). \end{aligned} \tag{5}$$

$\widehat{F}_{\text{timing}}$ and $\widehat{F}_{\text{power}}$ are GNN-based surrogates for TNS and total power, parameterized by θ_t and θ_p (Section 3.4). F_{wire} and F_{disp} model wirelength and displacement using refined analytical forms for smooth, scale-invariant gradients (Section 3.6), while F_{dens} applies the density penalty from DREAMPlace. Weights α, β, and γ balance these objectives, enabling HyPAS to adapt to diverse optimization goals.

We begin by introducing the continuous parameterization of discrete cell type candidates. Each cell instance i with candidate set $\mathcal{S}_i = \{s_{i,1}, s_{i,2}, \dots\}$ ($s_{i,k} = (d, v)$) is assigned a continuous, optimizable vector $\tilde{\boldsymbol{s}}_i \in \mathbb{R}^{|\mathcal{S}_i|}$, termed the `s-logits`, where each element $\tilde{s}_{i,k}$

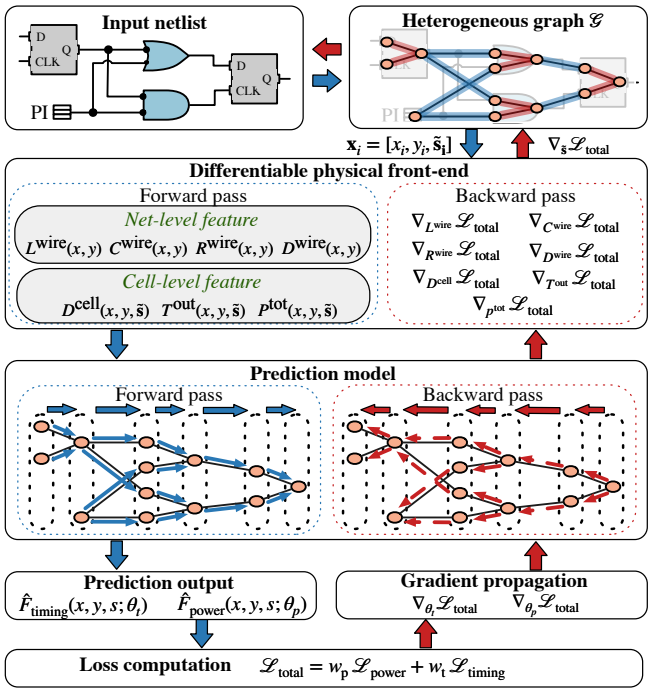


Figure 3: Differentiable model for timing and power.

corresponds to one candidate $s_{i,k} \in \mathcal{S}_i$. For training stability, the `S-logits` $\tilde{\boldsymbol{s}}_i$ are randomly initialized, with a positive bias (e.g., +2) toward the current cell type. The candidate set $\mathcal{S}_i$ is enumerated and zero-padded to a uniform length, with invalid entries masked by large negative values to ensure negligible probabilities after softmax:

$$p_{i,k} = \frac{\exp(\tilde{s}_{i,k}/\tau)}{\sum_j \exp(\tilde{s}_{i,j}/\tau)}, \quad \tau > 0 \text{ (temperature)}. \tag{6}$$

The forward pass selects the most probable candidate $s_i = \arg\max_k p_{i,k}$, while the STE allows gradients to flow through the `s-logits`.

3.4 Differentiable Timing and Power Modeling

To obtain $\widehat{F}_{\text{timing}}$ and $\widehat{F}_{\text{power}}$ in Equation (5), repeatedly calling external STA or power analysis tools incurs a high runtime cost. To enable gradient-based gate sizing, we require a differentiable timing/power engine that can support gradient flow. Recent studies [21, 22] show that GNN-based predictors can achieve ~**90%** accuracy with ~**1s** inference, offering fast and backpropagation-friendly guidance. Following [23], which formulates differentiable placement by converting cell locations into continuous features and propagating congestion penalties backward, we build a similar framework for timing and power (see Figure 3). It integrates a differentiable physical front-end with a GNN predictor, allowing gradients from high-level objectives to flow to individual cell parameters. In the current version, gradients are applied only to `s-logits` $\tilde{\boldsymbol{s}}$, while cell locations remain optimized by DREAMPlace.

Graph Representation. The circuit is modeled as a directed heterogeneous graph $\mathcal{G}$, where each node denotes a cell's input or output pin, and each edge represents an inter- or intra-cell connection. Each node i has input features $\mathbf{x}_i = [x_i, y_i, \tilde{\mathbf{s}}_i]$.

Differentiable Physical Front-End. To balance differentiability, accuracy, and efficiency, the physical front-end computes smooth

features at both the net and cell levels. For each connection $e = (u \to v)$, wire length is estimated using a differentiable Manhattan distance:

$$L^{\text{wire}}(x, y) = |x_u - x_v|_\tau + |y_u - y_v|_\tau, \ |z|_\tau = \tau \log\left(e^{z/\tau} + e^{-z/\tau}\right), \quad (7)$$

where $\tau > 0$ controls smoothness and $|z|_\tau$ approximates the absolute value for differentiability. Wire parameters scale linearly with length, with r_w and c_w denoting per-unit resistance and capacitance:

$$R^{\text{wire}}(x, y) = r_w L^{\text{wire}}(x, y), \ C^{\text{wire}}(x, y) = c_w L^{\text{wire}}(x, y). \quad (8)$$

Wire delay is estimated as:

$$D^{\text{wire}}(x, y) = R^{\text{drv}} \cdot C^{\text{wire}}(x, y) + 0.5 R^{\text{wire}}(x, y) \cdot C^{\text{wire}}(x, y), \quad (9)$$

where driver resistance R^{drv} is differentiable with respect to cell type. The total load capacitance C^{load}, computed as the sum of all downstream wire and input capacitances, remains fully differentiable.

To model cell-level behavior, we use pretrained MLPs that replace discrete NLDM tables. All quantities ($D^{\text{cell}}, T^{\text{out}}, P^{\text{tot}}$, etc.) depend on $(x, y, \tilde{s})$, but we omit these arguments for brevity. The delay model $\mathcal{F}_\omega$ predicts cell delay by: $[D^{\text{cell}}, T^{\text{out}}] = \mathcal{F}_\omega(T^{\text{in}}, C^{\text{load}})$, where T^{in} is the predefined input slew. The power model $\mathcal{F}_\psi$ estimates internal power as: $P^{\text{int}} = \mathcal{F}_\psi(T^{\text{out}}, C^{\text{load}})$. Total power combines switching, internal, and leakage components: $P^{\text{tot}} = \alpha C^{\text{load}} V_{\text{DD}}^2 f + P^{\text{int}} + P^{\text{leak}}$, where P^{leak} is the average leakage power from the cell library.

Timing and Power Message Passing. To capture timing and power characteristics, we employ two message-passing networks that share input features but have distinct parameters and objectives. The timing network propagates delay along directed edges and uses "max" aggregation to emphasize long-delay (critical) paths, consistent with static timing analysis. The power network adopts "mean" aggregation to capture average switching activity and energy flow. Each produces a graph-level embedding $\boldsymbol{g}_*$, which is mapped to a predicted cost

$$\widehat{F}_* = \phi_*(\boldsymbol{g}_*), \quad *\in \{\text{timing, power}\}, \quad (10)$$

where $\phi_*(\cdot)$ is a task-specific regression head.

Both predictors are trained independently with AdamW and early stopping. The total loss $\mathcal{L}_{\text{total}} = w_{\text{p}}\mathcal{L}_{\text{power}} + w_{\text{t}}\mathcal{L}_{\text{timing}}$ is a weighted combination of separate loss functions for power and timing.

3.5 STE-Based Gate Sizing Optimization

Discrete gate sizing is challenging due to its non-differentiable nature. To enable gradient-based learning, we adopt the straight-through estimator (STE) [24], which replaces the zero gradient $\frac{\partial q_i}{\partial z_i} = 0$ (where z_i is continuous and q_i its quantized counterpart) with $\frac{\partial \mathcal{L}}{\partial z_i} \approx \frac{\partial \mathcal{L}}{\partial q_i}$, allowing gradients to flow through discrete decisions. In our framework, gate sizing is formulated as a classification task parameterized by learnable `s-logits`: the forward pass selects the cell type with the highest softmax probability, while the STE enables end-to-end gradient propagation through this discrete choice.

As detailed in Algorithm 1, at each iteration t, each cell i is parameterized by learnable `s-logits` $\tilde{\boldsymbol{s}}_i^{(t)}$ associated with its discrete candidate set $\mathcal{S}_i$. Softmax probabilities $p_{i,k}$ are computed to form a continuous relaxation s_i^{soft} for gradient flow, while the hard selection s_i^{hard} chooses the highest-probability candidate for evaluation. To balance forward accuracy and backward differentiability, we define an STE variable s_i^{fwd} that equals s_i^{hard} in the forward pass, but passes gradients through s_i^{soft} during backpropagation.

Algorithm 1 STE-Based Sizing for Cell i at Iteration t

1: **Input:** `s-logits` $\tilde{\boldsymbol{s}}_i^{(t)}$, discrete candidate set $\mathcal{S}_i = \{s_{i,k}\}$;
2: **Output:** Updated `s-logits` $\tilde{\boldsymbol{s}}_i^{(t+1)}$, quantized cell type $s_i^{(t)}$;
3: Compute soft probabilities $p_{i,k}$ (Equation (6));
4: Obtain soft relaxation and hard selection:

$$s_i^{\text{soft}} = \sum_k p_{i,k} s_{i,k}, \quad s_i^{\text{hard}} = \mathcal{S}_{i,\arg\max_k p_{i,k}}.$$

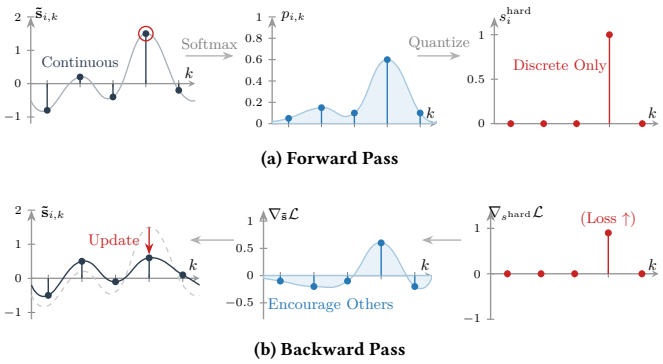
5: Construct STE forward variable:

$$s_i^{\text{fwd}} = s_i^{\text{hard}} - \text{detach}(s_i^{\text{soft}}) + s_i^{\text{soft}}.$$

6: Evaluate timing/power using $(x_i, y_i, s_i^{\text{fwd}})$ to obtain $\mathcal{L}_{\text{total}}$;
7: Backpropagate $\mathcal{L}_{\text{total}}$ to get gradients $\nabla_{\tilde{\boldsymbol{s}}_i^{(t)}} \mathcal{L}_{\text{total}}$;
8: Apply gradient clipping and update `s-logits`:

$$\tilde{\boldsymbol{s}}_i^{(t+1)} \leftarrow \tilde{\boldsymbol{s}}_i^{(t)} - \eta \cdot \text{clip}\left(\nabla_{\tilde{\boldsymbol{s}}_i^{(t)}} \mathcal{L}_{\text{total}}\right);$$

9: Quantize cell type for deployment: $s_i^{(t)} \leftarrow s_i^{\text{hard}}$;



(a) Forward Pass

(b) Backward Pass

Figure 4: Illustration of the STE process. (a) Forward pass maps continuous `s-logits` to a discrete selection s_i^{hard} (where k is the candidate index). (b) Backward pass propagates gradients to update `s-logits`, where the STE gradient distribution encourages alternative candidates to reduce loss.

The forward pass computes timing and power based on $(x_i, y_i, s_i^{\text{fwd}})$, and the overall loss $\mathcal{L}_{\text{total}}$ is backpropagated through the chain

$$\nabla_{\theta_t} \mathcal{L}_{\text{total}}, \nabla_{\theta_p} \mathcal{L}_{\text{total}} \to \nabla_{(x,y,\tilde{s})} \mathcal{L}_{\text{total}} \to \nabla_{\tilde{s}} \mathcal{L}_{\text{total}}. \quad (11)$$

Gradients on $(x, y, \tilde{s})$ are decomposed into wire-level ($L^{\text{wire}}, R^{\text{wire}}, D^{\text{wire}}$) and cell-level ($D^{\text{cell}}, P^{\text{tot}}$) components. Through STE, gradients on s propagate to $\tilde{\boldsymbol{s}}_i^{(t)}$, which are updated via gradient descent (learning rate η, with clipping). The quantized cell type $s_i^{(t)} = s_i^{\text{hard}}$ is finally deployed, completing one continuous-to-discrete update cycle.

Figure 4 visualizes the process. In the forward pass, the continuous `s-logits` $\tilde{\boldsymbol{s}}_i$ are mapped via softmax to probabilities $p_{i,k}$, and the highest-probability candidate is quantized to s_i^{hard} for evaluation. In the backward pass, the gradient $\nabla_s \mathcal{L}$ from the discrete choice is redistributed through STE to `s-logits` $\nabla_{\tilde{s}} \mathcal{L}$. The selected candidate receives a positive gradient (reducing its probability), while others receive negative gradients, steering $\tilde{\boldsymbol{s}}^{(t+1)}$ toward better options in the next iteration.

Table 1: Statistics and original PPA results of the benchmarks.

Benchmarks	#Nets	#FFs	#Cells	TNS (ps)	Power (W)	HPWL (μm)
des3	2420	128	2180	-24207.58	1.25E-02	7751333
des	2450	190	2317	-2049.19	1.58E-02	6335284
tv80s	3748	359	3730	-76722.98	1.10E-02	12400603
aes	4660	670	4393	-6833.96	4.34E-02	20396670
mc_top	5635	1065	5410	-66845.74	1.26E-02	18863912
ac97_top	7836	2191	7750	-198213.16	1.32E-01	22022564
ariane	10883	19894	105730	-3131.37	3.31E-02	754789504
aes_cipher[1]	11891	530	56194	-13971.28	7.86E-02	41497468
pci[2]	12264	3313	12091	-389996.69	1.28E-01	47203984
fpu	22707	660	22192	-5874.58	1.34E+02	54656912
NVpartition[3]	182553	47834	167833	-35175448	1.65E+00	1876563456
mempool[4]	206646	18167	182940	-11420379	1.29E+01	1033172480
netcard_fast	300220	84025	298384	-553493952	1.68E+00	3195323392

[1] aes_cipher_top [2] pci_bridge32 [3] NV_NVDLA_partition_c [4] mempool_tile_wrap

3.6 Differentiable Wirelength & Displacement

We adopt smooth analytical formulations for wirelength and displacement in Equation (5) to enable efficient gradient-based optimization. For $F_{\text{wire}}(\boldsymbol{x}, \boldsymbol{y})$, we define the surrogate wirelength $WL(\boldsymbol{x}, \boldsymbol{y})$ using two differentiable approximations of HPWL: log-sum-exp (LSE) and weighted-average (WA). LSE provides a convex surface for fast convergence, while WA better preserves local structure and topology. Both are evaluated during optimization, and the better result is retained to improve robustness. Instead of minimizing absolute HPWL, we optimize its normalized improvement relative to the initial layout:

$$F_{\text{wire}}(\boldsymbol{x}, \boldsymbol{y}) = \Delta_{\text{WL}} = \frac{WL^{(0)} - WL(\boldsymbol{x}, \boldsymbol{y})}{WL^{(0)}}, \tag{12}$$

where $WL^{(0)}$ is the initial surrogate wirelength. This baseline-referenced formulation ensures scale invariance and stabilizes gradients.

The displacement term $F_{\text{disp}}(\boldsymbol{x}, \boldsymbol{y})$ measures the average displacement from initial positions using a smooth L1-norm approximation $\sqrt{\Delta x^2 + \epsilon} + \sqrt{\Delta y^2 + \epsilon}$ to ensure differentiability during optimization.

3.7 Legalization and Detailed Placement

After differentiable refinement, the layout may slightly violate discrete constraints such as row alignment or site legality. We restore legality using OpenROAD legalization and detailed placement [20], which remove overlaps and align cells to valid sites per Equation (4). Finally, `check_placement` ensures design rule compliance.

4 Experimental Results

Our HyPAS framework operates in the 2025 ICCAD CAD Contest Docker environment [1]. Core algorithms are built on DREAMPlace and OpenROAD, with model training using DGL [25] and PyTorch [26]. Experiments are conducted on workstation with an Intel® Core™ i7-14700K CPU @ 5.6 GHz (20 cores), an NVIDIA® GeForce RTX 4090 GPU, and 128 GB RAM. Benchmarks come from the official contest suite [1], including public and hidden cases. Incremental placement optimization is performed using the ASAP7 PDK [27] with an NLDM timing model. Table 1 summarizes benchmark statistics and reports the original layout PPA as a reference.

4.1 Overall Performance

According to the formulation in Section 2.2, TNS, power, and HPWL are treated as the primary optimization objectives, with weighting factors fixed at $\alpha = \beta = \gamma = 1$ for consistency across evaluations. Displacement and runtime are also evaluated as complementary metrics to assess their influence on overall performance. Experimental results are summarized in Table 3, Table 4, and Figure 5. The post-optimization results demonstrate clear PPA improvements, reported as percentage improvements over the original input layouts. For fair comparison, reference data from the 1st, 2nd, and 3rd place teams of the 2025 ICCAD CAD Contest are used as baselines. Additional evaluations are performed on OpenROAD [5] (using `repair_timing` and legalization) and DREAMPlace4.0 [28].

Table 2: Comparison of total PPA improvement ($\alpha = \beta = \gamma = 1$).

Benchmarks	Total PPA Improvement					
	1st	2nd	3rd	OpenROAD	[28]	HyPAS
des3	-0.68%	**13.05%**	-0.20%	-23.73%	6.30%	11.77%
des	13.42%	13.36%	5.37%	-17.44%	4.69%	**18.53%**
tv80s	-0.58%	2.86%	1.23%	-6.93%	3.44%	**7.14%**
aes	**53.07%**	-27.32%	-44.38%	-32.86%	-18.31%	55.71%
mc_top	-5.60%	5.37%	-1.93%	-11.41%	7.89%	**18.13%**
ac97_top	39.46%	18.21%	18.48%	-12.79%	23.99%	**60.37%**
ariane	80.17%	82.57%	**85.72%**	53.35%	-3.60%	81.86%
aes_cipher[1]	48.08%	40.54%	-0.80%	-15.69%	13.35%	**48.91%**
pci[2]	54.25%	16.73%	36.57%	23.63%	18.89%	**71.01%**
fpu	65.46%	-4.72%	48.75%	46.89%	-69.81%	**66.29%**
NVpartition[3]	18.43%	**26.38%**	7.27%	1.69%	0.10%	24.73%
mempool[4]	-6.08%	**21.50%**	-0.33%	0	0	-0.79%
netcard_fast	40.86%	1.74%	-0.02%	0	-9.01%	**64.34%**
Average	30.79%	16.17%	11.98%	0.36%	-1.70%	**40.62%**

Experimental results show that HyPAS consistently delivers significant gains across all benchmarks. Specifically, the average improvements for TNS, power, and HPWL reach 24.95%, 12.45%, and 3.21%, respectively. Considering overall performance as the weighted sum of these improvements (with $\alpha = \beta = \gamma = 1$), HyPAS achieves an average improvement of 40.62%, outperforming the existing baselines—1st (30.79%), 2nd (16.17%), 3rd (11.98%), OpenROAD (0.36%), and DREAMPlace (-1.70%), as shown in Table 2.

While multi-objective optimization involves trade-offs, HyPAS maximizes overall gains under unified weights (α=β=γ=1) and delivers stable improvements across different design scales. For representative cases, HyPAS improves TNS by 58.36% on the small design "aes" with minor trade-offs in power (-6.91%) and HPWL (4.26%), while on the large-scale "NVpartition", it still achieves 31.85% TNS improvement and 0.13% HPWL reduction with only 7.27% power overhead. Moreover, compared with discrete frameworks (OpenROAD) and fully differentiable ones (DREAMPlace), our hybrid method consistently delivers superior results, confirming the necessity of integrating discrete and differentiable optimization.

Displacement and runtime are also evaluated for completeness. HyPAS achieves an average displacement of 3336.09 μm, indicating that moderate local cell movement during incremental placement can notably improve global PPA at a controllable cost. In terms of runtime, shorter execution time reflects higher search efficiency. Except for the 2nd place and DREAMPlace4.0 [28] baselines, HyPAS converges faster on most benchmarks while maintaining optimization quality, achieving a balance between performance and computational cost.

4.2 Adaptivity Analysis

To assess adaptability of HyPAS under varying weight factors (α, β, γ), we analyze the "aes" benchmark as shown in Figure 6. Six configurations are tested, each prioritizing one or two PPA objectives (TNS, power, HPWL). The total PPA improvement is computed as $\Delta_{\text{total}} = \alpha \cdot \Delta_{\text{TNS}} + \beta \cdot \Delta_{\text{power}} + \gamma \cdot \Delta_{\text{HPWL}}$. Unlike baselines that

Table 3: Comparison of TNS and total power improvements.

Benchmarks	TNS Improvement						Total Power Improvement					
	1st	2nd	3rd	OpenROAD	[28]	HyPAS	1st	2nd	3rd	OpenROAD	[28]	HyPAS
des3	0	**2.53%**	1.28%	-5.36%	-0.48%	-0.53%	0	**11.20%**	-0.80%	-9.60%	0	0.80%
des	9.52%	5.81%	5.42%	-13.43%	-1.58%	**18.76%**	0	**8.86%**	0	-1.90%	0	-5.70%
tv80s	0	**7.99%**	2.80%	-0.91%	-0.37%	0.23%	**0**	-4.55%	-0.91%	-3.64%	-0.91%	**0**
aes	**58.36%**	-30.26%	-51.39%	-20.79%	-21.77%	**58.36%**	-6.91%	**5.30%**	-2.30%	-11.29%	-0.23%	-6.91%
mc_top	0	8.60%	1.29%	-9.99%	3.90%	**9.17%**	0	**2.38%**	**2.38%**	-0.79%	0.79%	5.56%
ac97_top	-16.32%	**8.06%**	-7.12%	-7.80%	3.48%	-0.66%	55.15%	10.61%	26.06%	-3.03%	18.18%	**58.03%**
ariane	99.76%	91.47%	87.37%	54.37%	-3.64%	**99.79%**	-17.82%	-7.25%	**0**	-0.91%	**0**	-17.82%
aes_cipher[1]	**38.27%**	37.85%	0.43%	-9.66%	9.43%	**38.27%**	1.02%	**2.93%**	-0.89%	-4.45%	0.25%	1.02%
pci[2]	2.34%	13.52%	6.38%	17.13%	7.04%	**23.11%**	**49.22%**	3.91%	30.78%	7.03%	7.81%	44.14%
fpu	**70.77%**	-5.38%	50.41%	47.90%	-60.69%	**70.77%**	-4.48%	**1.49%**	-0.75%	-0.75%	-3.73%	-4.48%
NVpartition[3]	31.85%	30.07%	13.39%	1.70%	0.02%	**31.88%**	-7.27%	**2.42%**	0	0	0	-7.27%
mempool[4]	**1.22%**	-0.65%	0	0	0	-0.01%	-6.98%	**22.48%**	0	0	0	-0.78%
netcard_fast	-54.23%	**2.95%**	0	0	-10.07%	-24.76%	95.11%	-1.19%	0	0	-1.19%	**95.30%**
Average	18.58%	13.27%	8.48%	4.09%	-5.75%	**24.95%**	12.08%	4.51%	4.12%	-2.26%	1.61%	**12.45%**

Table 4: Comparison of HPWL improvements and average cell displacement.

Benchmarks	HPWL Improvement						Average Manhattan Displacement (μm)					
	1st	2nd	3rd	OpenROAD	[28]	HyPAS	1st	2nd	3rd	OpenROAD	[28]	HyPAS
des3	-0.68%	-0.68%	-0.68%	-8.77%	6.78%	**11.50%**	0	0	0	66.78	802.22	1736.69
des	3.90%	-1.31%	-0.05%	-2.11%	**6.27%**	5.47%	759.36	11.49	1654.33	17.48	714.54	3694.49
tv80s	-0.58%	-0.58%	-0.66%	-2.39%	4.72%	**6.92%**	0	0	3.45	21.69	776.79	1371.02
aes	1.63%	-2.36%	**9.31%**	-0.78%	3.69%	4.26%	893.71	0	3188.21	7.35	896.24	893.71
mc_top	-5.60%	-5.60%	-5.60%	-0.62%	3.19%	**3.41%**	0	0	0	4.64	799.59	1204.64
ac97_top	0.63%	-0.46%	-0.46%	-1.97%	2.33%	**3.01%**	733.93	0	0.01	5.49	753.73	25445.15
ariane	-1.77%	-1.65%	-1.65%	-0.11%	**0.05%**	-0.11%	0.72	0	0.58	0.88	0.85	4.49
aes_cipher[1]	8.80%	-0.23%	-0.34%	-1.58%	3.67%	**9.63%**	1173.32	0	4.31	19.84	713.86	1173.32
pci[2]	2.69%	-0.70%	-0.59%	-0.53%	**4.04%**	3.76%	915.95	7.49	0	2.60	944.28	5806.61
fpu	-0.84%	-0.84%	-0.92%	-0.27%	-5.39%	**0**	0	0	3.10	1.36	1716.26	0
NVpartition[3]	-6.15%	-6.11%	-6.12%	-0.01%	0.08%	**0.13%**	0.32	0	0.85	0.13	0.61	0.42
mempool[4]	-0.33%	-0.33%	-0.33%	0	**0.03%**	0	0	0	0	0	0.18	0.03
netcard_fast	-0.02%	-0.02%	-0.02%	0	**2.25%**	-6.20%	0	0	0	0	2028.19	2038.64
Average	0.13%	-1.61%	-0.62%	-1.47%	2.44%	**3.21%**	344.41	1.46	373.45	11.40	780.56	3336.09

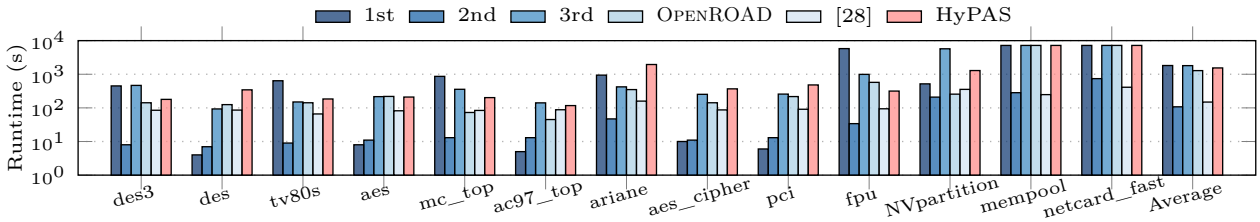


Figure 5: Runtime comparison (maximum time limit: 7200s).

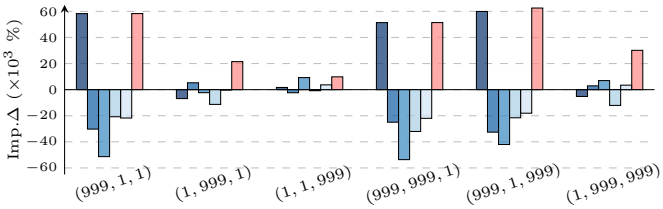


Figure 6: Total PPA improvement across (α, β, γ) variations.

lack adaptive multi-objective optimization and often degrade under changing weights, HyPAS consistently achieves superior results. For instance, when $(\alpha, \beta, \gamma) = (1, 999, 999)$—with power and HPWL heavily weighted—the total PPA improvement reaches 30132%, far exceeding all baselines (−5221%, 2906%, 6947%, −12077%, 3436%).

Across all six configurations, HyPAS averages a total improvement of 26262%, while baselines show much lower or even negative performance (17972%, −9250%, −15037%, −11161%, −6202%). These results demonstrate HyPAS's ability to balance multiple objectives, delivering higher overall performance and efficient convergence.

5 Conclusion

In summary, HyPAS introduces a hybrid framework combining discrete optimization with differentiable refinement for cell relocation, sizing, and buffering. By integrating process-aware sizing, STE-driven relaxation, and differentiable timing-power modeling within the OpenROAD and DREAMPlace, HyPAS unites rule-based robustness with gradient scalability. Experiments show significant PPA and runtime improvements, enabling sign-off quality and extensibility.

References

[1] Y.-C. Lu, R. Liang, W.-H. Liu, and H. Ren, "2025 iccad cad contest problem c: Incremental placement optimization beyond detailed placement: Simultaneous gate sizing, buffering, and cell relocation," 2025, iCCAD CAD Contest problem description.

[2] C. J. Alpert, S. K. Karandikar, Z. Li, G.-J. Nam, S. T. Quay, H. Ren, C. N. Sze, P. G. Villarrubia, and M. C. Yildiz, "Techniques for fast physical synthesis," *Proceedings of the IEEE*, vol. 95, no. 3, pp. 573–599, 2007.

[3] P. Spindler, U. Schlichtmann, and F. M. Johannes, "Kraftwerk2—a fast force-directed quadratic placement approach using an accurate net model," *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, vol. 27, no. 8, pp. 1398–1411, 2008.

[4] Y. Du, Z. Guo, Y. Lin, R. Wang, and R. Huang, "Fusion of global placement and gate sizing with differentiable optimization," in *Proceedings of the 43rd IEEE/ACM International Conference on Computer-Aided Design*, ser. ICCAD '24, 2024, pp. 1–9.

[5] T. Ajayi and D. Blaauw, "Openroad: Toward a self-driving, open-source digital layout implementation tool chain," in *Proceedings of Government Microcircuit Applications and Critical Technology Conference*, ser. GOMACTech '19, 2019.

[6] C. N. Sze, C. J. Alpert, J. Hu, and W. Shi, "Path based buffer insertion," in *Proceedings of the 42nd Annual Design Automation Conference*, ser. DAC '05, 2005, pp. 509–514.

[7] C. J. Alpert, M. Hrkić, J. Hu, and S. T. Quay, "Fast and flexible buffer trees that navigate the physical layout environment," in *Proceedings of the 41st Annual Design Automation Conference*, ser. DAC '04, 2004, pp. 24–29.

[8] A. Stefanidis, D. Mangiras, C. Nicopoulos, D. Chinnery, and G. Dimitrakopoulos, "Autonomous application of netlist transformations inside lagrangian relaxation-based optimization," *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, vol. 40, no. 8, pp. 1672–1686, 2020.

[9] Y. Ye, P. Xu, L. Ren, T. Chen, H. Yan, B. Yu, and L. Shi, "Learning-driven physically aware large-scale circuit gate sizing," *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, vol. 44, no. 5, pp. 1901–1914, 2024.

[10] Y. Lin, S. Dhar, W. Li, H. Ren, B. Khailany, and D. Z. Pan, "Dreamplace: Deep learning toolkit-enabled gpu acceleration for modern vlsi placement," in *Proceedings of the 56th Annual Design Automation Conference 2019*, ser. DAC '19, 2019, pp. 1–6.

[11] L. Liu, B. Fu, M. D. F. Wong, and E. F. Y. Young, "Xplace: An extremely fast and extensible global placement framework," in *Proceedings of the 59th ACM/IEEE Design Automation Conference*, ser. DAC '22, 2022, pp. 1309–1314.

[12] Y.-C. Lu, Z. Guo, K. Kunal, R. Liang, and H. Ren, "Insta: An ultra-fast, differentiable, statistical static timing analysis engine for industrial physical design applications," in *Proceedings of the 62nd ACM/IEEE Design Automation Conference*, ser. DAC '25. IEEE, 2025, pp. 1–7.

[13] Z. Guo and Y. Lin, "Differentiable-timing-driven global placement," in *Proceedings of the 59th ACM/IEEE Design Automation Conference*, ser. DAC '22, 2022, pp. 1315–1320.

[14] B. Fu, L. Liu, M. D. F. Wong, and E. F. Y. Young, "Hybrid modeling and weighting for timing-driven placement with efficient calibration," in *Proceedings of the 43rd IEEE/ACM International Conference on Computer-Aided Design*, ser. ICCAD '24, 2024, pp. 1–9.

[15] Y. Shi, S. Xu, S. Kai, X. Lin, K. Xue, M. Yuan, and C. Qian, "Timing-driven global placement by efficient critical path extraction," in *Proceedings of the Design, Automation and Test in Europe Conference*, ser. DATE '25. IEEE, 2025, pp. 1–7.

[16] W. Ding, Z. Zhang, G. He, and P. Cao, "A physical and timing aware placement optimization framework based on graph neural network," in *Proceedings of the 43rd IEEE/ACM International Conference on Computer-Aided Design*, ser. ICCAD '24, 2024, pp. 1–9.

[17] Y. Du, Z. Guo, Y. Hsu, Z. Xiong, S. Kim, D. Z. Pan, R. Wang, and Y. Lin, "Addressing continuity and expressivity limitations in differentiable physical optimization: A case study in gate sizing," in *Proceedings of the International Symposium on Electronics Design Automation*, ser. ISEDA '25. IEEE, 2025, pp. 375–380.

[18] T. Luo, D. Newmark, and D. Z. Pan, "Total power optimization combining placement, sizing and multi-vt through slack distribution management," in *Proceedings of the Asia and South Pacific Design Automation Conference*, ser. ASP-DAC '08. IEEE, 2008, pp. 352–357.

[19] C.-P. Chen, C. C. N. Chu, and M. D. F. Wong, "Fast and exact simultaneous gate and wire sizing by lagrangian relaxation," in *Proceedings of the 1998 IEEE/ACM International Conference on Computer-Aided Design*, ser. ICCAD '98, 1998, pp. 617–624.

[20] S. Do, M. Woo, and S. Kang, "Fence-region-aware mixed-height standard cell legalization," in *Proceedings of the Great Lakes Symposium on VLSI*, ser. GLSVLSI '19, 2019, pp. 259–262.

[21] Z. Guo, M. Liu, J. Gu, S. Zhang, D. Z. Pan, and Y. Lin, "A timing engine inspired graph neural network model for pre-routing slack prediction," in *Proceedings of the 59th ACM/IEEE Design Automation Conference*, ser. DAC '22, 2022, pp. 1207–1212.

[22] Z. Wang, S. Liu, Y. Pu, S. Chen, T.-Y. Ho, and B. Yu, "Restructure-tolerant timing prediction via multimodal fusion," in *Proceedings of the 60th ACM/IEEE Design Automation Conference*, ser. DAC '23. IEEE, 2023, pp. 1–6.

[23] S. Liu, Q. Sun, P. Liao, Y. Lin, and B. Yu, "Global placement with deep learning-enabled explicit routability optimization," in *Proceedings of the Design, Automation and Test in Europe Conference and Exhibition*, ser. DATE '21. IEEE, 2021, pp. 1821–1824.

[24] P. Yin, J. Lyu, S. Zhang, S. J. Osher, Y. Qi, and J. Xin, "Understanding straight-through estimator in training activation quantized neural nets," in *Proceedings of the International Conference on Learning Representations*, ser. ICLR '19, 2019.

[25] M. Wang *et al.*, "Deep graph library: Towards efficient and scalable deep learning on graphs," 2019, iCLR workshop on representation learning on graphs and manifolds.

[26] A. Paszke, S. Gross, F. Massa, A. Lerer, J. Bradbury, G. Chanan, T. Killeen, Z. Lin, N. Gimelshein, L. Antiga *et al.*, "Pytorch: An imperative style, high-performance deep learning library," *Advances in Neural Information Processing Systems*, vol. 32, 2019.

[27] L. T. Clark, V. Vashishtha, L. Shifren, A. Gujja, S. Sinha, B. Cline, C. Ramamurthy, and G. Yeric, "Asap7: A 7-nm finfet predictive process design kit," *Microelectronics Journal*, vol. 53, pp. 105–115, 2016.

[28] P. Liao, D. Guo, Z. Guo, S. Liu, Y. Lin, and B. Yu, "Dreamplace 4.0: Timing-driven placement with momentum-based net weighting and lagrangian-based refinement," *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, vol. 42, no. 10, pp. 3374–3387, 2023.