C3PO: Commercial-Quality Global Placement via Coherent, Concurrent Timing, Routability, and Wirelength Optimization

Yi-Chen Lu, Hao-Hsiang Hsiao, Rongjian Liang, Wen-Hao Liu, and Haoxing Ren
NVIDIA Research
{yilu, haoxingr}@nvidia.com

***Abstract*—Despite achieving orders-of-magnitude runtime speedup, GPU-accelerated placers (GPU-Placers) still have extremely limited industrial adoption, largely due to the wide gaps in Power, Performance, and Area (PPA) metrics compared to those well-established CPU-centric commercial Physical Design (PD) tools. To overcome this issue, we introduce C3PO, the first commercial-quality, differentiable, multi-objective global placer that performs concurrent timing, routability, and wirelength optimization in a coherent manner with custom CUDA kernels. Particularly, we propose a convex-based framework that dynamically computes objective weights at each placement iteration by solving a quadratic problem, eliminating the need of manual parameter tuning. In the experiments, we rigorously validate C3PO with an industry-leading commercial PD tool and demonstrate that on 8 designs from TILOS [1] and IWLS [2] in ASAP $7nm$ [3], C3PO consistently outperforms the commercial tool by up to 16.7% in routed wirelength and 19.6% in switching power with complete full-flow validation.**

I. Introduction

With the relentless pursuit of high-performance and low-power designs, global placement remains an extremely critical challenge in modern Physical Design (PD) as cell locations determined at this stage dictate resistances and capacitances (RC) of interconnects, which directly impacts end-of-flow Power, Performance, and Area (PPA) metrics. Nonetheless, with the ever-increasing design complexity pushed by Moore's Law, the runtime of commercial global placers have drastically inflated to maintain solution quality. To overcome runtime bottlenecks of traditional CPU-centric placers, DREAMPlace [4] introduced the first differentiable, GPU-accelerated placer (GPU-Placer) leveraging modern Machine Learning (ML) infrastructure PyTorch [5]. Recently, several GPU-Placers were proposed to further optimize timing [6–11] or routability [12–16] beyond simple wirelength reduction. However, they exhibit fundamental shortcomings:

- **Timing.** Most works still weight nets or arcs heuristically, yielding noisy gradients and requiring delicate loss and parameter tuning with routing being ignored. Hence, timing gains vanish once designs are routed in the consideration of congestion.
- **Routability.** Most works rely on cell-inflation-based heuristics for optimization, which is harmful to wirelegnth as white space will be created. Other ML-based approaches differentiate through learned surrogates that are design-specific and non-generalizable.

In this paper, we address all the limitations aforementioned in commercial tools and state-of-the-art GPU-Placers by presenting C3PO, the first commercial-quality GPU-Placer that performs concurrent and coherent timing, routability, and wirelength optimization at each placement iteration. Particularly, we introduce novel custom CUDA kernels that compute exact gradients of global timing metrics including Total Negative Slack (TNS) and Worst Negative Slack (WNS), and the widely-adopted routability score RUDY [17], making each individual objective fully differentiable with respect to cell locations. Furthermore, we introduce an Multi-Gradient Descent Algorithm (MGDA) [18]-inspired objective weighting framework that balances the weight of each individual objective by systematically solving a quadratic program at each placement iteration. To rigorously validate the effectiveness and practicality of C3PO, we perform a head-to-head comparison against the global placer of an industry-leading commercial PD tool, as illustrated in Figure 1, where we directly replace the commercial tool's global placement engine with C3PO while strictly following the identical downstream flow for fair evaluation. Our main contributions are as follows:

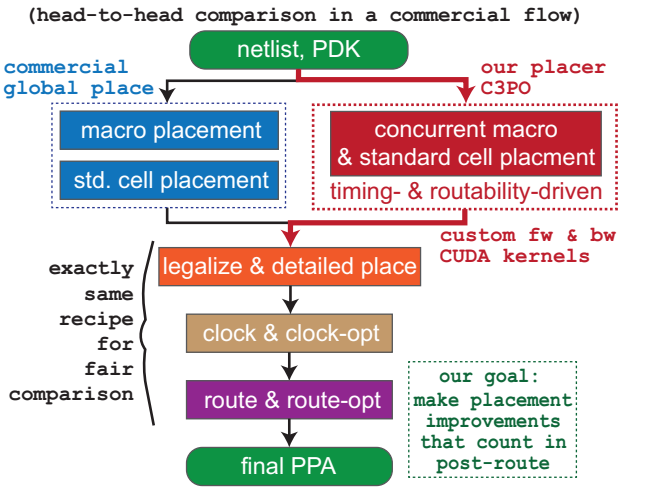

Fig. 1: Commercial validation flow between an industry-leading commercial placer and C3PO. Note that placement improvements are meaningless without post-route justification. Our goal is to make placement improvements that actually count in post-route.

- We present C3PO, the **first** commercial-quality, GPU-accelerated global placer that concurrently optimizes timing, routability, and wirelength in a GPU-accelerated and differentiable manner.
- We develop custom CUDA kernels enabling exact gradient computations of global timing metrics (TNS, WNS) and the RUDY-based routability score with respect to cell locations.
- We propose a convex-based framework that systematically determines objective weights at each placement iteration by solving a quadratic problem, which is design-agnostic and eliminates the need of parameter tuning.
- We perform direct comparisons with state-of-the-art timing-driven GPU-Placers [9–11] and demonstrate that C3PO not only achieves 31.2% better TNS but also 55% better routability score.
- We conduct head-to-head full-flow comparisons against an industry-leading commercial PD tool by replacing the entire global placement stage and maintaining identical downstream optimization recipes. Experimental results on IWLS [2] and TILOS [1] benchmarks in ASAP $7nm$ [3] demonstrate that C3PO achieves significant and consistent end-of-flow PPA improvements. On ARIANE136 (TILOS), we observe C3PO outperforms the tool by 8.3% in routed wirelength and 8.7% in switching power.

TABLE I: Key differences of recent works on timing-driven GPU-Placers. Red elements denote better scenarios.

Feature	DREAMPlace 4.0 [6]	DAC'22 [9]	ICCAD'24 [8]	DATE'25 [11]	DAC'25 [10]	C3PO (ours)
Native STA	no (OpenTimer)	Yes	no (OpenTimer)	no (OpenTimer)	prop. only	yes
STA platform	CPU	GPU	CPU + GPU	CPU	CPU + GPU	GPU
STA Style*	calibration	differentiable	calibration	calibration	calib. + diff.	differentiable
Statistical STA	no	no	no	no	yes (POCV)	yes (POCV)
Cell Delay Calc.	interpolation	interpolation	interpolation	interpolation	interpolation	solver-based
Weight Scaling	momentum	linear	momentum	momentum	linear	convex-based
Update Method†	net-based	graph-based	arc-based	arc-based	arc-based	graph-based
Update Metric	slack	gradient	slack	slack	gradient	gradient
Routability Aware	no	no	no	no	no	yes (differentiable kernel)

* "differentiable" denotes whether TNS/WNS are truly end-to-end differentiable with respect to cell locations, which is the preferred case.
† "graph-based" indicates the gradient is computed using the entire timing graph, whereas "arc-based" approaches do not account for slew effects.

II. Related Work

A. GPU-Accelerated Timing-Driven Global Placement

Recently, GPU-accelerated Timing-Driven Placement (TPD) methods have emerged to address the runtime barrier of traditional CPU-based TDP by embedding timing information into vanilla GPU-Placers [4, 19] that solely focus on wirelength reduction. Particularly, several works have improved upon DREAMPlace [4] by incorporating momentum-driven net-weighting [6, 20], arc-weighting [7, 10, 11], or differentiable STA [9] into global placement iterations. Nonetheless, despite significant runtime advantages over CPU-based methods, these GPU-based TDP placers have critical limitations preventing industrial-grade PPA performance, primarily due to:

i) **Net-level weighting.** With a single weight to an entire net [6, 20] blurs the boundary between critical and non-critical paths, producing gradients that steer placement in the wrong direction.
ii) **Arc-level weighting.** Per-arc scaling [7, 10, 11] is finer, yet it ignores the shared-RC coupling between sinks; a "non-critical" branch can still drive slew and invalidate the estimate.
iii) **Heuristic gradients.** Both net- and arc-based schemes recompute weights every iteration, add runtime, and deliver only approximate $\nabla_x\{\text{WNS,TNS}\}$, limiting optimisation fidelity.
iv) **Academic "differentiable" STA.** The method in [9] provides analytic gradients but relies on bilinear NLDM interpolation; extreme load–slew pairs common in early placement collapse to table corners, eroding accuracy.
v) **Routing blind spot.** All published GPU-TDP studies ICCAD'15 benchmarks, assuming congestion-free "virtual" routing. Timing gains vanish once real routers introduce detours.

In this work, we overcome all these limitations above by presenting C3PO, a differentiable multi-objective GPU-Placer that truly meets industrial standard. Table I summarizes the key distinctions between our engine C3PO and state-of-the-art GPU-based TDP frameworks.

B. GPU-Accelerated Routability-Driven Global Placement

Routability-driven placement (RDP) has become increasingly critical over the past decade, driven by escalating congestion challenges in modern industrial designs [21]. In existing GPU-Placers, both DREAMPlace [4] and Xplace [22] adopt the classical cell inflation paradigm to optimize RUDY-based [17] routability metrics. Although this simple technique can improve routability, it often comes at a significant cost in wirelength, as the added whitespace undermines density targets and disrupts placement quality. Recently, ML-based data-driven approaches [12, 13, 15] leverage predictive models for congestion estimation or end-of-flow cell density distribution to provide early routability guidance during placement. However, these ML models inherently suffer from inaccuracies and poor generalization across diverse designs and technology nodes, which severely limits their real-world applications. In this work, we fundamentally address and surpass the limitations of both cell-inflation-based and ML-driven routability methods. Particularly, we present the first fully differentiable, GPU-accelerated routability optimization kernel, which computes exact gradients of the DREAMPlace routability metric with respect to cell locations. In the experiments, we demonstrate that our routability optimization kernel outperforms the predominant cell inflation in literature in both wirelength and routability metrics.

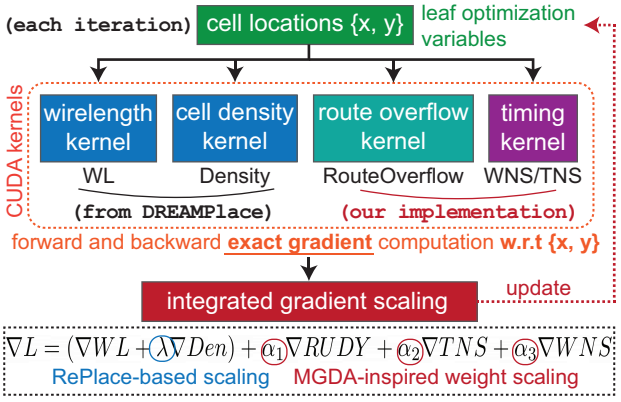

Fig. 2: Overview of the exact, differentiable multi-objective optimization in C3PO. A convex-based weight scaling technique is introduced to automatically determine objective weights at each iteration.

III. Algorithm

A. Overview of C3PO

Figure 2 presents an overview of our differentiable, multi-objective GPU placer, C3PO, which concurrently optimizes critical PPA metrics, including timing (TNS/WNS), routability, and wirelength, via rigorous exact gradient computations. Unlike existing GPU-based placers that heavily depend on indirect net- or arc-based weighting schemes for timing optimization and cell inflation techniques for routability improvement, we introduce custom CUDA kernels that directly compute global timing and routability objectives and their precise gradients for accurate cell location updates in addition to DREAMPlace's original wirelength and density kernels. To further enhance placement stability and quality, we propose an MGDA-inspired gradient scaling strategy that dynamically balances each individual objective at each iteration, which eliminates the need for manual parameter tuning and prevents placement divergence.

B. Our Differentiable STA Engine (Timing Kernels)

The design of our differentiable STA engine is inspired by the philosophy of INSTA [10], which splits STA into two stages:

(i) **Delay computation**, where we derive closed-form, analytically differentiable expressions for net and cell delays;
(ii) **Timing propagation**, which accumulates computed delays over the timing graph to derive global timing metrics (WNS/TNS)

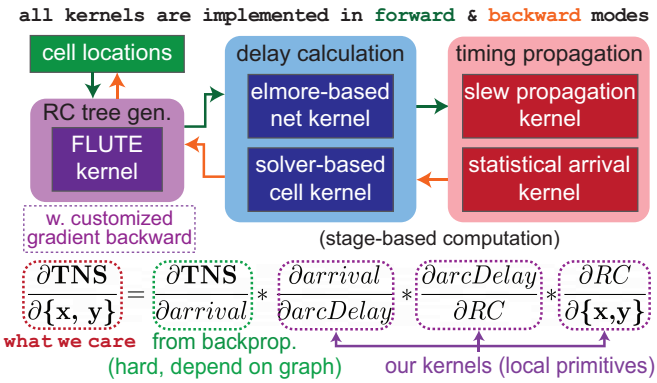

Fig. 3: Overview of our timing kernels for end-to-end, differentiable TNS and WNS computation with respect to cell locations. With the backpropagation feature in PyTorch [5], we calculate exact timing gradients to cell locations with custom CUDA kernels for local computation.

and back-propagates gradients to cell coordinates.

However, unlike INSTA which relies on external third-party tools for delay computation, our STA engine fully integrates differentiable delay calculations directly into GPU kernels, with cell locations explicitly serving as leaf optimization variables. This self-contained approach significantly enhances optimization precision, runtime efficiency, and stability, setting it apart from prior GPU-accelerated TDP works as summarized in Table I. Figure 3 gives an overview of the key components in our timing kernel, which includes a custom FLUTE-based kernel for Steiner point insertion and backward gradient distribution, an Elmore-based kernel for net delay computation, a solver-based kernel for cell delay calculation, and propagation kernels for graph-based timing analysis (GBA mode). Below we describe each component of our differentiable STA engine.

1) Gradient Distribution of Steiner Points

Steiner point insertion critically affects interconnect length and routing quality. In C3PO, Steiner points are determined by FLUTE [23], a widely-adopted technique for Rectilinear Steiner Minimal Tree (RSMT) construction. Although FLUTE's topology-insertion step is non-differentiable, once Steiner nodes are placed, we can differentiate subsequent RC-tree calculations with respect to node coordinates. In this work, unlike the prior approach of periodically adjusting Steiner coordinates with gradient updates [9], we propose a gradient redistribution method of Steiner points, which pushes each Steiner point's gradient to its surrounding non-Steiner nodes, including those on diagonals. Concretely, let ∇x_s and ∇y_s denote the partial derivatives of some objective (e.g., Elmore delay) with respect to the Steiner node s. For each neighboring node n, define:

$$\Delta x_n = w_x(n,s)\mathrm{sgn}(x_s - x_n)\left|\nabla x_s\right|, \tag{1}$$

$$\Delta y_n = w_y(n,s)\mathrm{sgn}(y_s - y_n)\left|\nabla y_s\right|, \tag{2}$$

where $\mathrm{sgn}(\cdot)$ is the sign function (e.g., $\mathrm{sgn}(\alpha) = +1$ if $\alpha \geq 0$ and -1 otherwise), and $w_x(n,s)$, $w_y(n,s)$ are nonzero only if n is horizontally, vertically, or diagonally aligned with s. Specifically:

- Horizontal neighbors receive Δx_n, pulling them along the x-axis.
- Vertical neighbors receive Δy_n, pulling them along the y-axis.
- Diagonal neighbors receive both Δx_n and Δy_n.

Intuitively, if a neighbor n is to the left of a Steiner node s ($x_n < x_s$), it will be pushed to the right (positive Δx_n), whereas if n is to the right ($x_n > x_s$), it will receive a negative shift, bringing n closer to s. The key idea of this approach is to guide cell placement towards the direction of RC minimization, which outperforms the simplistic Steiner nodes' gradient adjustments in [9] as such update can easily worsen RC. Note we periodically re-compute RC-trees with FLUTE to adapt the topology as cell locations evolve.

2) Differentiable Elmore Net Delay Kernel

Following [8, 9], we adopt the Elmore model during placement. Each net's RC tree is evaluated in a single CUDA thread via alternating post-order DFS and pre-order BFS sweeps:

$$L_u = C_u + \sum_{v\in \mathrm{ch}(u)} L_v, \qquad D_v = D_u + R_{uv}L_v, \tag{3}$$

$$LD_u = C_u D_u + \sum_{v\in \mathrm{ch}(u)} LD_v, \qquad \beta_v = \beta_u + R_{uv}LD_v, \tag{4}$$

$$I_v = \sqrt{2\beta_v - D_v^2}, \qquad S_v = \sqrt{S_u^2 + I_v^2}. \tag{5}$$

Here u is the parent of v; R_{uv} and C_u grow linearly with Manhattan wire length, so every term is differentiable w.r.t. cell coordinates. Finally, reverse sweeps compute gradients: first a post-order DFS accumulates ∂I, then a pre-order BFS propagates ∂L.

3) Solver-Based Cell Delay/Slew Calculation

With the effective load ℓ and input slew s from the Elmore kernel, we query each cell-arc's timing tables. Academic tools typically use bilinear interpolation between the four surrounding points [9, 10, 24]. During early placement, however, cells may be scattered or stacked, producing extreme (ℓ, s) values that lie outside the grid; interpolation then collapses to a corner lookup, yielding highly inaccurate delays and misleading gradients. C3PO avoids this pitfall by adopting the solver-based evaluation employed in Synopsys PrimeTime [25], which stays accurate for arbitrary load-slew pairs.

Particularly, our **solver-based** method follows three steps:

(i) Bracket search. Given queries ℓ and s, we locate indices as:

$$L_i \leq \ell \leq L_{i+1}, \qquad S_j \leq s \leq S_{j+1}, \tag{6}$$

giving the four NLDM values $f_{ij}, f_{i+1j}, f_{ij+1}, f_{i+1j+1}$. Out-of-range queries clamp to table corners.

(ii) Polynomial fit. Solve for

$$Z(\ell,s) = A + B\ell + Cs + D\ell s, \tag{7}$$

$$Z(L_i,S_j) = f_{ij} \quad Z(L_{i+1},S_j) = f_{i+1j}, \tag{8}$$

$$Z(L_i,S_{j+1}) = f_{ij+1} \quad Z(L_{i+1},S_{j+1}) = f_{i+1j+1}. \tag{9}$$

which uniquely determines the colored coefficients A, B, C, and D and captures higher-order load–slew coupling.

(iii) Query and gradients. Delay or slew and its derivatives follow directly:

$$Z(\ell,s) = \mathrm{CellArcDelay/Slew}(\ell,s), \tag{10}$$

$$\frac{\partial Z}{\partial \ell} = B + Ds, \qquad \frac{\partial Z}{\partial s} = C + D\ell. \tag{11}$$

This entire procedure runs in a custom CUDA kernel and, combined with our differentiable Elmore model, produces delay/slew estimates following commercial practice.

4) Differentiable Timing Propagation

Thus far, we have detailed the delay calculation kernels of our differentiable STA engine, covering both Elmore kernels for net delays and solver-based kernels for cell delays. The last piece of our STA engine is timing propagation, where arrival times and slew values are propagated across the timing graph, ultimately deriving global metrics such as WNS and TNS. Building directly on INSTA [10], we view every pin arrival as a Gaussian $\mathcal{N}(\mu,\sigma^2)$. Along a single arc $(i \to j)$ the statistics simply add: $\mu_j = \mu_i + \mu_{ij}$, $\sigma_j^2 = \sigma_i^2 + \sigma_{ij}^2$. Each CUDA thread therefore updates (μ,σ) in one forward pass. At a multi-fanin node we need a differentiable "latest-arrival" operator. Replacing the hard max with the max-normalized Log-Sum-Exp yields

$$\mathrm{LSE}(\mathbf{A}) = M + \tau \ln \sum_k \exp\big((A_k - M)/\tau\big), \quad M = \max_k A_k, \tag{12}$$

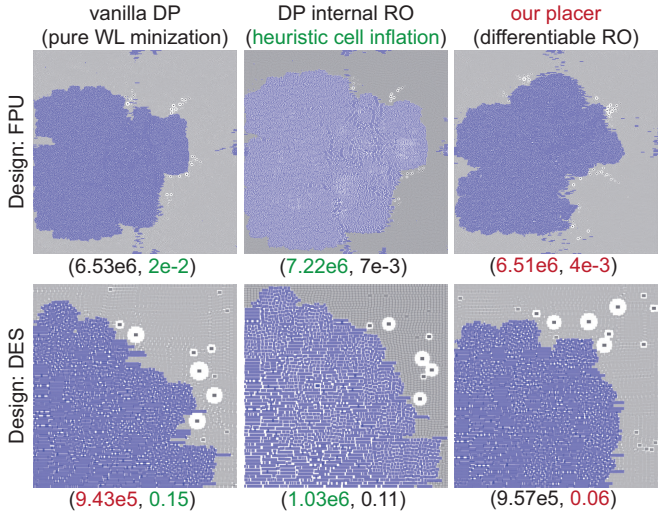

Fig. 4: Comparisons of final $(\mathrm{HPWL}, \mathrm{RouteOverflow})$ between DREAMPlace [4] with cell-inflation technique and C3PO. Our method yields the highest routability score (measured by DREAMPlace) while incurring minimal wirelength penalty.

whose gradient is the usual softmax. With a tiny temperature ($\tau = 0.1$) the result is numerically stable yet converges to max as $\tau \to 0$, giving smooth, placement-aware timing curves. Combining the Elmore kernels with this soft-merge pass produces an *end-to-end differentiable STA* that delivers exact gradients of WNS/TNS w.r.t. cell coordinates, enabling sub-second timing optimization that surpasses prior TDP placers in both accuracy and scale.

C. Differentiable Routability Kernel

Until now, RUDY [17] remains a reliable analytic proxy for routing demand, yet the only published gradient derivation [12] is not exact. In this paper, we give complete derivation of RUDY gradients, all of which are implemented with custom CUDA kernels:

Bounding box and spans. For a net n whose pins are $\{(x_p, y_p)\}$, $x_{\min,n} = \min_p x_p$, $x_{\max,n} = \max_p x_p$ and $y_{\min,n}, y_{\max,n}$ are defined analogously. Padding with a small ε avoids zero width:

$$x_{\mathrm{span},n} = (x_{\max,n} - x_{\min,n}) + \varepsilon,\ y_{\mathrm{span},n} = (y_{\max,n} - y_{\min,n}) + \varepsilon. \tag{13}$$

Per–bin usage. Let bin (i,j) span $[x_i^\ell, x_i^h] \times [y_j^\ell, y_j^h]$. The planar overlap is defined as:

$$\begin{aligned}\mathrm{overlap}_n(i,j) = \ & \max\Big(0,\ \min(x_{\max,n}, x_i^h) - \max(x_{\min,n}, x_i^\ell)\Big) \\ & \times \max\Big(0,\ \min(y_{\max,n}, y_j^h) - \max(y_{\min,n}, y_j^\ell)\Big).\end{aligned} \tag{14}$$

With net weight w_n, we can derive:

$$H_n(i,j) = \frac{w_n \,\mathrm{ov}_n}{y_{\mathrm{span},n}},\ V_n(i,j) = \frac{w_n \,\mathrm{ov}_n}{x_{\mathrm{span},n}},\ \mathrm{RUDY}_n = H_n + V_n. \tag{15}$$

Exact gradients. Using the chain and quotient rules we obtain, for a pin coordinate x_p,

$$\frac{\partial \mathrm{RUDY}_n}{\partial x_p} = \frac{w_n}{y_{\mathrm{span},n}} \frac{\partial \mathrm{ov}_n}{\partial x_p} + \frac{w_n}{x_{\mathrm{span},n}^2} \left[x_{\mathrm{span},n} \frac{\partial \mathrm{ov}_n}{\partial x_p} - \mathrm{ov}_n \frac{\partial x_{\mathrm{span},n}}{\partial x_p} \right]. \tag{16}$$

Here $\partial \mathrm{ov}_n / \partial x_p = \pm y_{\mathrm{span},n}$ if moving x_p shifts the left $(-)$ or right $(+)$ edge of the box, otherwise 0; and $\partial x_{\mathrm{span},n} / \partial x_p = \pm 1$ for those edges. If several pins share an edge (e.g. multiple sinks at $x_{\max,n}$), the edge's gradient is divided equally among them, following the subgradient of $\max(\cdot)$. Derivatives w.r.t. y_p are analogous.

Figure 4 compares our differentiable routability kernel with vanilla DREAMPlace [4] and the widely used cell-inflation heuristic. Cell inflation reduces congestion but enlarges wirelength by inserting whitespace. Our approach lowers RouteOverflow by up to 45.4 % relative to it with negligible HPWL overhead.

D. MGDA-Based Objective Weighting in C3PO

C3PO optimizes wirelength and density simultaneously with timing (TNS, WNS) and routability (RouteOverflow) at every global placement iteration (Figure 2). Unlike existing GPU-TDP/RDP placers tuning objective weights with heuristics, in this work, we devise an MGDA-based objective scaling strategy. A classical MGDA problem solves $\{w\}$ in the following equation:

$$\min_{\{w_i \geq 0,\ \sum w_i = 1\}} \Big\| \sum_{i=0}^{K-1} w_i g_i \Big\|^2, \tag{17}$$

where g_i is the gradient of objective i. However, in placement, directly solving (17) might cause secondary goals (timing, routability) to conflict with the primary objectives, leading to placement divergence. Hence, we recast the MGDA formulation as follows:

Relaxed scheme

We fix $g_0 \equiv g_{\mathrm{WL+DEN}}$ (scaled as in RePlAce [26]) and seek non-negative weights for the remaining gradients:

$$g_{\mathrm{comb}} = g_0 + \sum_{i=1}^{K-1} \alpha_i g_i. \tag{18}$$

To keep gradient updates at each placement iteration stable, we set:

$$\alpha_i = \omega_i^{(0)} \frac{\|g_0\|}{\|g_i\| + \varepsilon}, \qquad i = 1, \ldots, K-1, \tag{19}$$

where $\omega_i^{(0)}$ encodes user emphasis and $\varepsilon > 0$ avoids division by zero. Next, we leverage geometric insights of gradients by constructing the following matrices and vectors to encode interactions:

$$H_{ij} = g_i^T g_j, \quad i,j = 1, \ldots, K-1, \tag{20}$$

$$b_i = g_0^T g_i, \qquad i = 1, \ldots, K-1. \tag{21}$$

We then formulate a regularized linear system to dynamically compute the optimal secondary weights α_i at each iteration:

$$\big(H + \lambda I\big)\alpha \ = \ -b + \lambda \beta, \tag{22}$$

where $\lambda > 0$ controls the anchoring strength toward the initial guidance vector β. The closed-form solution to this system is:

$$\alpha^* \ = \ \big(H + \lambda I\big)^{-1} \big(-b + \lambda \beta\big). \tag{23}$$

Finally, to maintain numerical stability and physical meaningfulness, we enforce non-negativity constraints by clamping:

$$\alpha_i \ = \ \max\{\alpha_i^*, 0\}, \quad i = 1, \ldots, K-1 \tag{24}$$

The outcome of our effort is a smooth and scalable weighting scheme that eliminates the need of manual parameter tuning while ensuring placement convergence and effective PPA optimization.

IV. Experimental Results

In the experiments, we validate C3PO against an industry-leading commercial PD tool as well as several state-of-the-art academic GPU-based timing-driven placement (TDP) methodologies [9–11]. To thoroughly evaluate our framework, we leverage 9 representative designs selected from the well-established IWLS [2] and TILOS [1] benchmark suites, which encompass a diverse set of designs from standard-cell-only to macro-heavy ones. The goal of this work is to introduce a robust GPU-Placer that performs commercial-quality, concurrent, concurrent multi-objective placement optimization which can be adopted in modern industrial design flows. Finally, C3PO is implemented with PyTorch 2.1 and CUDA 11.8, where the timing and routability kernels are implemented with PyTorch's C++/CUDA extension. All evaluations are conducted on a high-performance

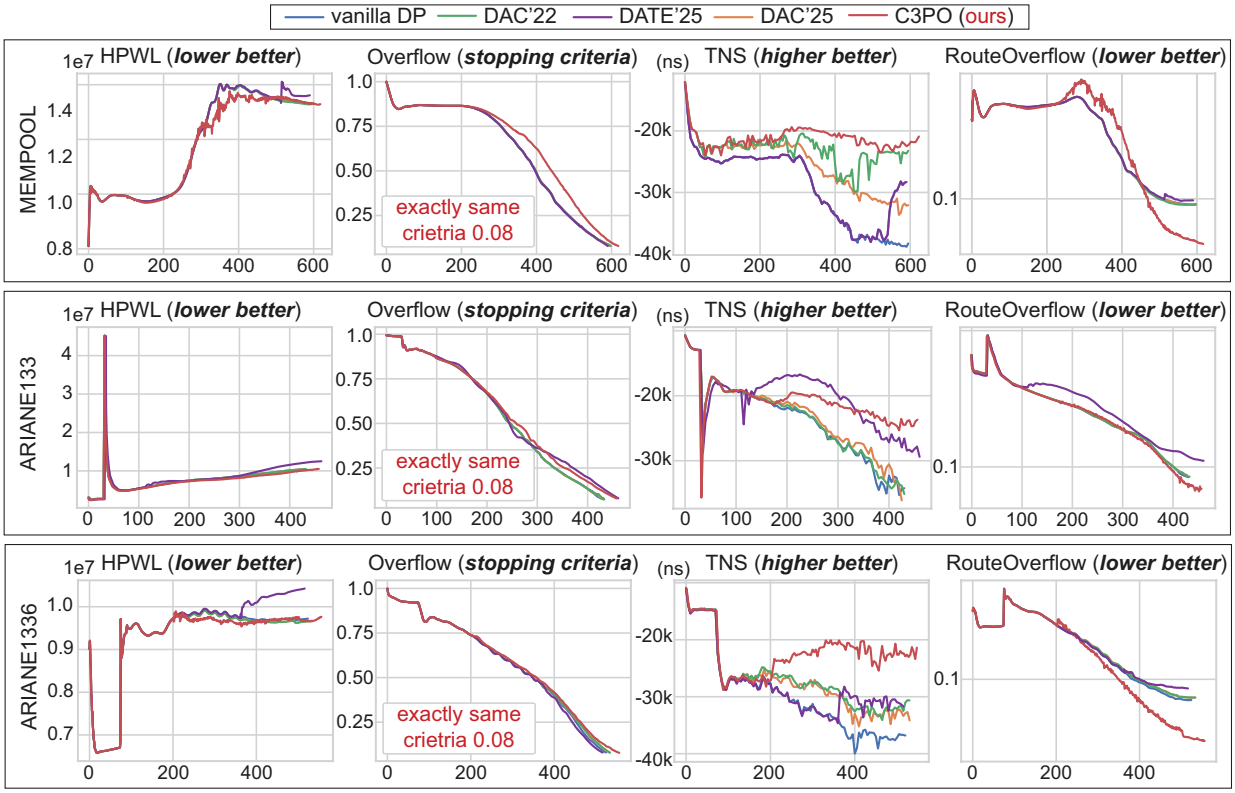

Fig. 5: Comparisons between C3PO and state-of-the-art GPU-based TDP works including DAC'22 [9], DATE'25 [11], and DAC'25 [10] on three renowned TILOS benchmarks [1]: MEMPOOL, ARIANE133, and ARIANE136. For DAC'22 [9], we reproduce the work to the best of our knowledge. For DATE'25 [11] and DAC'25 [10], we perform the validation using the open-source code. Finally, for C3PO, we perform concurrent timing and routability optimization as described in the Algorithm section. **All methods have the exact same stopping criteria of 0.08 overflow.**

TABLE II: Runtime (second) comparison of a timing update among state-of-the-art GPU-based TDP. Refer to Figure 6 for complete metrics.

Benchmark	DAC'22 [9]	DATE'25 [11]	DAC'25 [10]	C3PO
MEMPOOL	6.08	717.3	35.7	6.19
ARIANE133	4.51	120.6	31.1	4.95
ARIANE136	4.02	88.0	20.9	4.22

* we include Steiner point insertion runtime for [9] and C3PO.
† we use 8 threads in OpenTimer for [11] and [10].

Linux platform with a single NVIDIA A100 GPU (96GB memory) and an AMD EPYC 7742 processor (64 cores) with 2TB of RAM.

A. Comparisons with Academic GPU-based TDP Works

Figure 5 shows the comparisons between DREAMPlace [4], state-of-the-art academic GPU-based TDP works [9–11], and our placer C3PO at each placement iteration on 3 renowned TILOS [1] designs with macros synthesized in the ASAP7 [3] technology. It is shown that C3PO consistently achieves the best TNS and routability score across each benchmark compared to other state-of-the-art TDP methods with minimum overhead in HPWL compared with DREAMPlace. Note that for fair comparison, all global placement runs utilize the exactly same stopping criteria based on Overflow (i.e., cell density), which is a standard practice. Finally, the runtime comparison for timing updates is presented in Table II, clearly highlighting the substantial benefits of utilizing a native GPU-accelerated STA engine. Specifically, on the MEMPOOL benchmark, C3PO and DAC'22 [9] achieve speedups exceeding 100× compared to DATE'25 [11], and approximately 7× compared to DAC'25 [10].

Academic TDP Results Discussion. The experimental results presented in Figure 5 confirm the limitations we have identified in Table I and discussed in Section II-A. Specifically, the arc-based TDP method proposed in [11], while effective at optimizing timing, does so at the great cost of wirelength and routability, as clearly demonstrated by design ARIANE136. Moreover, we observe severe runtime degradation when applying [11] to logically complex designs such as MEMPOOL, where multiple endpoints share substantial overlapping fan-in cones. In such scenarios, even a single iteration of timing evaluation requires over 10 minutes, despite leveraging the highly parallelized `report_timing_topk` implementation introduced in [11]. As for DAC'25 [10], which employs a timing-update scheme similar to [11], achieves only marginal timing improvements, although it does not incur the significant wirelength penalties observed in [11]. The differentiable placement method in [9] exhibits comparatively smaller timing gains than C3PO. We attribute this difference to the observation that routability optimization inherently supports timing improvement, since reducing net bounding boxes through routability-driven gradient updates often results in reduced RC delays, beneficially influencing timing metrics. These empirical insights strongly support our central hypothesis: that concurrent and coherent optimization of timing and routability within a differentiable framework is essential for achieving optimal placement quality.

B. Comparisons with an Industry-Leading PD Tool

Table III presents a comprehensive, full-flow head-to-head comparison between C3PO and an industry-leading commercial PD tool, where we directly replace the entire global placement stage with our framework while maintaining identical downstream recipes as illustrated in Figure 2. It is shown that C3PO consistently achieves significant end-of-flow PPA improvements over all benchmarks, most notably we demonstrate up to 8.3% wirelength reduction for macro-based TILOS designs, and 16.7% improvement for standard-cell-only designs. These remarkable wirelength gains come with

TABLE III: **Key Optimization Results.** Full-flow head-to-head PPA comparisons between an industry-leading commercial PD tool and C3PO where C3PO replaced the entire global placement stage, including macro placement (if any), of the commercial tool. For designs with macros, C3PO performs concurrent macro and standard cell placement. **Note that all tools go through the exact same PD recipe after global placement and we use the exact same seed across all experiments to remove non-deterministic run-to-run variation.**

design ($7nm$) (# cells)	PD stage	industry-leading commercial PD tool							C3PO (ours, directly replace tool's global place)						
		routed WL (um)	WNS (ns)	TNS (ns)	# ep vios	internal (mW)	switching (mW)	cell area (um^2)	routed WL (um)	WNS (ns)	TNS (ns)	# ep vios	internal (mW)	switching (mW)	cell area (um^2)
ARIANE133 (117k cells)	place-opt	930622	-0.014	-0.327	168	453.3	26.2	16068.9	816641	-0.004	-0.054	41	451.0	24.1	15749.2
	clock-opt	949833	0.0	0.0	0	460.9	41.2	16077.1	868539.8	0.0	0.0	0	458.3	38.28	15788.9
(133 macros)	route-opt	1003527	-0.003	-0.009	7	452.6	40.80	16127.2	921253(-8.3%)	-0.003	-0.004	2	450.2	39.15	15856.6
ARIANE136 (113k cells)	place-opt	898275	-0.009	-0.986	477	462.8	22.9	16069.4	801139	-0.036	-5.818	1338	461.7	20.8	15736.2
	clock-opt	919735	0.0	0.0	0	451.6	41.2	16179.0	817556	-0.033	-1.170	212	450.5	39.4	15664.3
(136 macros)	route-opt	970236	-0.008	-0.035	14	461.3	40.06	16321.9	891853(-8.0%)	-0.007	-0.101	38	458.6	36.58	15964.2
MEMPOOL (162k cells)	place-opt	1177113	-1.208	-12.8k	19.1k	24705.4	9399.7	20829.7	1139195	-1.206	-12.9k	19.1k	24668.9	9264.2	20739.0
	clock-opt	1282571	-1.168	-12.4k	17.6k	25111.9	12225.8	22806.9	1228725	-1.157	-12.6k	19.1k	25042.1	12036.9	22442.4
(20 macros)	route-opt	1400162	-1.193	-12.2k	18.1k	26090.8	12762.3	24659.1	1346411 (-3.8%)	-1.191	-12.4k	19.2k	26077.2	12678.8	24312.6
mc_top (6.1k cells)	place-opt	22651	-0.173	-78.1	582	9.76	2.82	760.2	20759	-0.175	-93.1	777	9.25	2.31	676.9
	clock-opt	23198	-0.181	-83.1	600	9.79	5.11	783.78	21949	-0.181	-80.9	714	9.22	4.58	688.2
	route-opt	23161	-0.181	-79.1	559	9.88	4.99	801.1	21458(-7.4%)	-0.180	-77.3	674	9.33	4.50	702.3
aes_cipher (12.5k cells)	place-opt	43107	-0.094	-26.8	480	8.36	8.83	1161.6	42920	-0.089	-25.4	449	8.04	8.71	1149.7
	clock-opt	45893	-0.117	-30.8	517	8.28	10.23	1170.6	44845	-0.104	-29.3	513	8.25	10.06	1162.0
	route-opt	48671	-0.117	-27.2	502	8.69	10.09	1188.0	47034(-3.3%)	-0.102	-26.5	479	8.41	9.96	1181.1
i2c_master (663 cells)	place-opt	1501	-0.091	-4.13	82	2.33	0.65	89.2	1457	-0.066	-3.66	84	2.30	0.63	88.8
	clock-opt	1678	-0.104	-4.79	84	2.33	1.23	90.3	1635	-0.104	-4.26	83	2.30	1.21	89.4
	route-opt	1715	-0.123	-4.83	84	2.32	1.22	91.8	1638(-4.5%)	-0.087	-3.94	83	2.34	1.20	91.4
FPU (28.5k cells)	place-opt	76817	-0.929	-27.7	35	3.65	4.62	2640.9	66066	-0.968	-29.1	35	2.74	3.81	2325.4
	clock-opt	83832	-1.013	-30.6	36	4.30	5.19	2916.7	71009	-0.995	-29.9	36	3.00	4.16	2490.3
	route-opt	84129	-0.947	-28.1	35	4.21	5.01	2920.3	71536(-16.7%)	-0.950	-27.7	34	3.01	4.03	2495.2
DES (4.1k cells)	place-opt	12846	-0.112	-6.06	66	5.20	6.35	390.9	12666	-0.098	-5.84	65	6.22	5.53	332.6
	clock-opt	13559	-0.107	-6.50	66	6.66	7.00	414.3	12994	-0.095	-5.83	64	5.69	6.30	353.1
	route-opt	13429	-0.102	-6.21	66	6.79	6.72	421.1	12836(-4.4%)	-0.093	-5.24	64	5.87	6.07	363.2

Runtime Note: C3PO delivers 10-30x faster global placement than the commercial tool with similar full-flow runtime on the exact same P&R recipe.

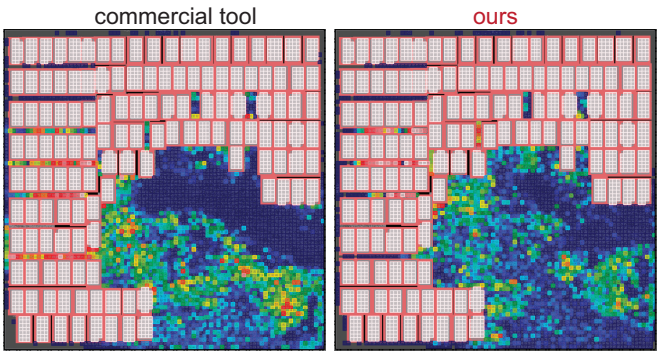

Fig. 6: Congestion comparison between an industry-leading commercial tool vs. C3PO on ARIANE136. Our routability optimization kernel improves a key congestion score by 68.3%.

either comparable timing performance or only marginal degradations, which translate well into dynamic power reduction. We note that AutoDMP [27] also compares its results to the TILOS benchmark. However, unlike AutoDMP heavily relying on extensive and impractical parameter sweeps, C3PO consistently delivers superior PPA results through its elegant convex-based objective weighting scheme, which simultaneously and dynamically optimizes critical PPA metrics without manual intervention, entirely eliminating the tedious and error-prone parameter tuning procedure. Finally, to evaluate our differentiable routability kernel, we leverage the exact same floorplan of ARIANE136, and perform head-to-head comparison between commercial tool's global placer and C3PO. Figure 6 highlights the promising results that our kernel reduces the hotspot reported by the commercial tool by 68.3% relative to the tool's own global placement.

Industrial Results Discussion. Besides the fact that modern commercial placers still cannot efficiently handle designs with excessive macros (e.g., ARIANE133 and ARIANE136) as described in [1, 27], we attribute the exceptional success of C3PO to our differentiable, gradient-driven, multi-objective global placement methodology thanks to the power of modern ML infrastructure. Particularly, by leveraging PyTorch [5], C3PO can coherently manage complex, multi-objective gradients at scale to update leaf optimization variables (e.g., cell locations) efficiently, which is a capability fundamentally absent from traditional CPU-based PD tools. We believe that beyond obvious runtime advantages, our gradient-guided, coherent global placement paradigm clearly shows profound opportunity to significantly enhance existing industrial CPU-centric PD methodologies.

V. Conclusion

We have presented C3PO, a GPU-accelerated, differentiable multi-objective global placer that concurrently optimizes timing, routability, and wirelength in a unified and coherent manner through dynamic, convex-based weight scaling. To address critical shortcomings in existing GPU-based TDP methods, we introduce a fully differentiable STA engine, featuring a solver-based CUDA kernel for industry-standard cell delay computation. Furthermore, we develop the first differentiable routability optimization kernel that computes exact gradients of routability metrics with respect to cell locations. Comprehensive experiments validated on a diverse set of designs demonstrate that C3PO consistently outperforms state-of-the-art academic GPU-based TDP methods in both timing and routability without degrading HPWL. Notably, compared with an industry-leading commercial PD tool, C3PO demonstrates much better end-of-flow PPA metrics by replacing the entire global placement step. We envision C3PO to transform existing CPU-centric commercial global placers into GPU-accelerated and differentiable paradigms. Inspired by the recent success of GPU-accelerated and machine learning [28–44], in design automation, we plan to enhance C3PO with similar technologies to further improve its performance and capabilities.

REFERENCES

[1] C.-K. Cheng, A. B. Kahng, S. Kundu, Y. Wang, and Z. Wang, "Assessment of reinforcement learning for macro placement," in Proceedings of the 2023 International Symposium on Physical Design.

[2] C. Albrecht, "Iwls 2005 benchmarks," in International Workshop for Logic Synthesis (IWLS), vol. 9, 2005.

[3] 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.

[4] Y. Lin et al., "Dreamplace: Deep learning toolkit-enabled gpu acceleration for modern vlsi placement," IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems.

[5] 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, 2019.

[6] P. Liao et al., "Dreamplace 4.0: Timing-driven global placement with momentum-based net weighting," in 2022 Design, Automation & Test in Europe Conference & Exhibition (DATE).

[7] W. Li et al., "Calibration-based differentiable timing optimization in non-linear global placement," in Proceedings of the 2024 International Symposium on Physical Design.

[8] B. Fu et al., "Hybrid modeling and weighting for timing-driven placement with efficient calibration," in Proceedings of the 43rd IEEE/ACM International Conference on Computer-Aided Design, 2024.

[9] Z. Guo and Y. Lin, "Differentiable-timing-driven global placement," in Proceedings of the 59th ACM/IEEE Design Automation Conference.

[10] 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 2025 62th ACM/IEEE Design Automation Conference (DAC), 2025.

[11] Y. Shi et al., "Timing-driven global placement by efficient critical path extraction," in Proceedings Design, Automation and Test in Europe Conference and Exhibition, IEEE, 2025.

[12] S. Liu, Q. Sun, P. Liao, Y. Lin, and B. Yu, "Global placement with deep learning-enabled explicit routability optimization," in 2021 Design, Automation & Test in Europe Conference & Exhibition (DATE).

[13] A. Agnesina et al., "Goalplace: Begin with the end in mind," in Proceedings of the 2025 ISPD, 2025.

[14] S. Zheng, L. Zou, S. Liu, Y. Lin, B. Yu, and M. Wong, "Mitigating distribution shift for congestion optimization in global placement," in 2023 60th ACM/IEEE Design Automation Conference (DAC).

[15] Y.-C. Lu, H. Ren, H.-H. Hsiao, and S. K. Lim, "Gan-place: Advancing open source placers to commercial-quality using generative adversarial networks and transfer learning," ACM Transactions on Design Automation of Electronic Systems, vol. 29, no. 2, pp. 1–17, 2024.

[16] Y.-C. Lu, H. Ren, H.-H. Hsiao, and S. K. Lim, "Dream-gan: Advancing dreamplace towards commercial-quality using generative adversarial learning," in Proceedings of the 2023 International Symposium on Physical Design, ISPD '23, (New York, NY, USA), p. 141–148, Association for Computing Machinery, 2023.

[17] P. Spindler and F. M. Johannes, "Fast and accurate routing demand estimation for efficient routability-driven placement," in 2007 Design, Automation & Test in Europe Conference & Exhibition.

[18] J.-A. Désidéri, "Multiple-gradient descent algorithm (mgda) for multiobjective optimization," Comptes Rendus Mathematique, 2012.

[19] L. Liu, B. Fu, M. D. Wong, and E. F. Young, "Xplace: An extremely fast and extensible global placement framework," in Proceedings of the 59th ACM/IEEE Design Automation Conference.

[20] J.-M. Lin, Y.-Y. Chang, and W.-L. Huang, "Timing-driven analytical placement according to expected cell distribution range," in Proceedings of the 2024 International Symposium on Physical Design.

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

[22] L. Liu et al., "Xplace: An extremely fast and extensible placement framework," IEEE TCAD.

[23] C. Chu and Y.-C. Wong, "Flute: Fast lookup table based rectilinear steiner minimal tree algorithm for vlsi design," IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems, 2007.

[24] T.-W. Huang and M. D. Wong, "Opentimer: A high-performance timing analysis tool," in 2015 IEEE/ACM International Conference on Computer-Aided Design (ICCAD), pp. 895–902, IEEE, 2015.

[25] "Primetime user guide: Advanced timing analysis," V-2023.03, Synopsys online Documentation, 2023.

[26] C.-K. Cheng et al., "Replace: Advancing solution quality and routability validation in global placement," IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems, vol. 38, no. 9, pp. 1717–1730, 2018.

[27] A. Agnesina et al., "Autodmp: Automated dreamplace-based macro placement," in Proceedings of the 2023 International Symposium on Physical Design.

[28] Y.-C. Lu, S. Pentapati, and S. K. Lim, "Vlsi placement optimization using graph neural networks,"

[29] Y.-C. Lu, S. Nath, V. Khandelwal, and S. K. Lim, "Rl-sizer: Vlsi gate sizing for timing optimization using deep reinforcement learning," in 2021 58th ACM/IEEE Design Automation Conference (DAC), pp. 733–738, IEEE, 2021.

[30] H.-H. Hsiao, Y.-C. Lu, P. Vanna-Iampikul, A. Agnesina, R. Liang, Y.-H. Lu, H. Ren, and S. K. Lim, "Dco-3d: Differentiable congestion optimization in 3d ics," in 2025 62nd ACM/IEEE Design Automation Conference (DAC), pp. 1–7, IEEE, 2025.

[31] Y.-C. Lu, W.-T. Chan, D. Guo, S. Kundu, V. Khandelwal, and S. K. Lim, "Rl-ccd: Concurrent clock and data optimization using attention-based self-supervised reinforcement learning," in 2023 60th ACM/IEEE Design Automation Conference (DAC), pp. 1–6, 2023.

[32] Y.-C. Lu, S. S. K. Pentapati, L. Zhu, K. Samadi, and S. K. Lim, "Tp-gnn: A graph neural network framework for tier partitioning in monolithic 3d ics," in 2020 57th ACM/IEEE Design Automation Conference (DAC).

[33] Z. Yang et al., "Ml-based fine-grained modeling of dc current crowding in power delivery tsvs for face-to-face 3d ics," in Proceedings of the 2025 International Symposium on Physical Design.

[34] Y.-C. Lu, S. Pentapati, L. Zhu, G. Murali, K. Samadi, and S. K. Lim, "A machine learning-powered tier partitioning methodology for monolithic 3-d ics," IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems, vol. 41, no. 11, pp. 4575–4586, 2021.

[35] H.-H. Hsiao, P. Vanna-Iampikul, Y.-C. Lu, and S. K. Lim, "Ml-based physical design parameter optimization for 3d ics: From parameter selection to optimization," in Proceedings of the 61st ACM/IEEE Design Automation Conference, pp. 1–6, 2024.

[36] Y.-C. Lu, W.-T. Chan, V. Khandelwal, and S. K. Lim, "Driving early physical synthesis exploration through end-of-flow total power prediction," in Proceedings of the 2022 ACM/IEEE Workshop on Machine Learning for CAD, pp. 97–102, 2022.

[37] H.-H. Hsiao, Y.-C. Lu, S. K. Lim, and H. Ren, "Buffalo: Ppa-configurable, llm-based buffer tree generation via group relative policy optimization," in Proceedings of the 44th IEEE/ACM International Conference on Computer-Aided Design (ICCAD), 2025.

[38] Y.-C. Lu, S. Nath, S. S. K. Pentapati, and S. K. Lim, "A fast learning-driven signoff power optimization framework. In 2020 IEEE," in ACM International Conference On Computer Aided Design (ICCAD), pp. 1–9, 2020.

[39] Y.-C. Lu, S. Nath, V. Khandelwal, and S. K. Lim, "Doomed run prediction in physical design by exploiting sequential flow and graph learning," in 2021 IEEE/ACM International Conference On Computer Aided Design (ICCAD), pp. 1–9, IEEE.

[40] Y.-H. Lu, H.-H. Hsiao, Y.-C. Lu, H. Ren, and S. K. Lim, "Differentiable tier assignment for timing and congestion-aware routing in 3d ics," in 2026 31st ACM/IEEE Asia South Pacific Design Automation Conference (ASP-DAC), 2026.

[41] Y.-C. Lu, S. Pentapati, and S. K. Lim, "The law of attraction: Affinity-aware placement optimization using graph neural networks," in Proceedings of the 2021 International Symposium on Physical Design, pp. 7–14, 2021.

[42] H.-H. Hsiao, Y.-C. Lu, P. Vanna-Iampikul, and S. K. Lim, "Fasttuner: Transferable physical design parameter optimization using fast reinforcement learning," in Proceedings of the 2024 International Symposium on Physical Design, pp. 93–101, 2024.

[43] Y.-H. Chung, S. Jiang, W.-L. Lee, Y. Zhang, H. Ren, T.-Y. Ho, and T.-W. Huang, "Simpart: A simple yet effective replication-aided partitioning algorithm for logic simulation on gpu," in European Conference on Parallel Processing, pp. 197–210, Springer, 2025.

[44] Y.-H. Chung, N. Oh, M. G. Balabhadra Naga Venkata, A. Shiledar, S. Kundu, V. Khandelwal, and T.-W. Huang, "Accelerating gate sizing using gpu," in European Conference on Parallel Processing, 2025.