LATTE: Legality-Assured Differentiable Timing-Driven Detailed Placement

Jing Mai[1,3,†], Yi-Chen Lu[4], Yibo Lin[2,3], Haoxing Ren[4]
{[1]School of Computer Science, [2]School of Integrated Circuits}, Peking University, Beijing, China
[3]Institute of Electronic Design Automation, Peking University, Wuxi, China
[4]NVIDIA Corporation
jingmai@pku.edu.cn, yilu@nvidia.com, yibolin@pku.edu.cn, haoxingr@nvidia.com

Abstract

Differentiable techniques for timing optimization have become the dominant paradigm in global placement, surpassing weight-tuning heuristics by directly leveraging gradients that encompass full-chip context. However, in detailed placement, state-of-the-art flows still rely on rule-based or enumeration-centric methods that ignore existing gradient landscape, sacrificing significant wirelength only for marginal timing gains. To overcome this issue, we present LATTE, the first legality-assured, end-to-end differentiable timing-driven detailed placer that fuses continuous gradient smoothness with detailed discrete granularity. Particularly, at each iteration, LATTE relaxes local density constraints around selected timing-critical cells, steering relocation from exact timing and wirelength gradients with step sizes in annealed schedules, and enforces end-of-place legality via incremental legalization and bad-move filtering. Across input placements from mainstream commercial and academic placers, LATTE consistently delivers significant timing improvement with minimal routed wirelength impact while ensuring legal final solutions. On 15 designs in a 7*nm* technology node, LATTE on average improves state-of-the-art timing-driven detailed placers including DREAMPlace-4.0 TCAD and Rsyn by 29.7% in TNS and 44.4% in routed wirelength, with all metrics verified by an industry-leading commercial tool.

1 Introduction

Detailed placement (DP) is recognized as the grand challenge in modern Physical Design (PD) [1, 2] as today's commercial flows typically perform a single Global Placement (GP) pass to set an initial layout, then devote the majority of placement runtime to incremental optimization for Power, Performance, and Area (PPA) recovery [2]. With the relentless pursuit of high-performance chips, timing has become the prime objective in industrial placement optimization, which exhibits tightly-coupled trade-offs among wirelength, congestion, and power that must be navigated through legality-aware relocation.

Although timing-driven GP has entered a new era with differentiable techniques [3, 4], state-of-the-art DP engines [5–7] remain local and enumeration-centric, which completely ignores the continuous global objective landscape achieved in GP and reduces the relocation search to trial-and-error loops with heuristic scoring. Particularly, these methods [5–10] explore new candidate locations in a manner that neither aligns with true global timing metrics such as Total Negative Slack (TNS) and Worst Negative Slack (WNS) nor accounts for legalization and routability, which is stemmed from their following fundamental drawbacks:

- **Local sensitivities misaligned with global timing.** Relocation decisions are driven by window-level or per-arc proxies of local delay improvements which ignore full-graph propagation impact and therefore correlate poorly with global timing metrics.
- **Slack-driven cell selection is flawed.** Critical cells are selected for relocation by slack. However, unlike gradients, slack neither provides guidance on the change of TNS/WNS from a relocation nor indicates a direction for movement.

† This work was conducted during an internship at NVIDIA.

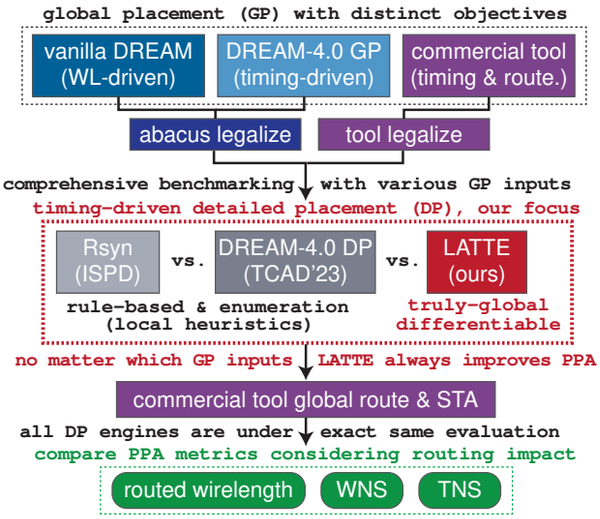

Figure 1: Comprehensive evaluation flow of LATTE. Given any GP input, we perform head-to-head comparisons between LATTE and state-of-the-art detailed placers: Rsyn and DREAMPlace-4.0 TCAD, using a commercial tool for strict PPA verification with routing impact.

- **Enumeration-centric, incoherent optimization.** Moves are generated by enumeration and accepted using local Δtiming and Δwirelength thresholds, which treat objectives as separate checks rather than a single coherent update, leading to convergence issues and sub-optimal outcomes.

In this paper, we overcome all the above drawbacks of previous works by presenting LATTE, the first legality-assured, end-to-end differentiable timing-driven detailed placer. LATTE introduces differentiable optimization into detailed placement by unifying continuous gradient-based updates with discrete legality constraints. Particularly, it computes exact timing gradients that capture full-graph propagation impact, optimizes timing and wirelength simultaneously in one descent step, and preserves legality through in-the-loop incremental timing-aware legalization, all achieved with custom CUDA kernels.

As shown in Fig. 1, we rigorously evaluate LATTE against prior state-of-the-art timing-driven detailed placers, including DREAMPlace-4.0 TCAD [7] and Rsyn [5, 6]. Given diverse legalized GP inputs from mainstream academic placers and an industry-leading commercial tool, LATTE on average delivers **29.7%** improvement in TNS and **44.4%** reduction in wirelength across 15 designs from well-established benchmark suites IWLS [11] and TILOS [12] compared with previous works [5–7] (all verified by the commercial tool after global routing). The goal of this work is to demonstrate that in detailed placement, gradient-guided optimization provides a principled and much more superior alternative to existing ad-hoc, discrete heuristics that have long dominated this regime. Our main contributions are as follows:

- We introduce LATTE, the first-ever legality-assured, end-to-end differentiable, timing-driven detailed placer that unifies continuous gradient-based relocation with discrete legality constraints using custom GPU-accelerated CUDA kernels.
- At each round, LATTE computes exact timing gradients of TNS and WNS with respect to each cell's location, select critical cells with a balanced independent set matching algorithm, and then

DAC '26, Long Beach, CA, USA

ACM ISBN 979-8-4007-2254-7/2026/07
https://doi.org/10.1145/3770743.3803960

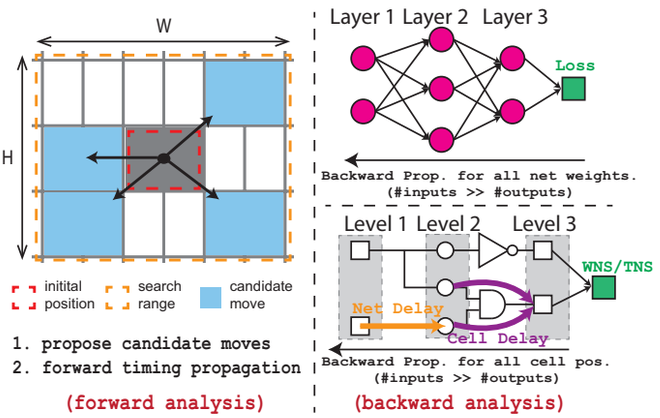

Figure 2: Left: Lagrangian-based methods [7, 13] conduct forward sensitivity analysis. They use Lagrangian weights to indirectly estimate the TNS impact of local delay changes from candidate moves. Right: Backward analysis, analogous to backpropagation in neural network training (as shown in the top-right corner of this illustration), simultaneously computes explicit descent directions for all cell positions. It leverages the high input-to-output ratio of STA for efficient computation.

performs coherent descent updates that simultaneously optimize timing and wirelength with step sizes in annealed schedules.

- We maintain legality via in-the-loop, timing-driven incremental legalization coupled with a lightweight bad-move filter, ensuring an overlap-free, routability-friendly state at every iteration.
- Across all experiments, we rigorously perform validation with an industry-leading commercial tool (name withheld under license). Specifically, after DP, we run commercial global routing to capture congestion impact and report timing and routed wirelength. Across 15 designs in a 7nm technology node, LATTE decisively outperforms prior state-of-the-art detailed placers [5–7].

2 Preliminary

2.1 Timing-Driven Detailed Placement

Given a netlist $\mathcal{G} = (V,E)$ and an initial placement solution $(\boldsymbol{x}^{(0)},\boldsymbol{y}^{(0)})$, the primary objective of timing-driven detailed placement is to produce a cell displacement $(\delta\boldsymbol{x},\delta\boldsymbol{y})$ such that the new placement $(\boldsymbol{x}^{(0)}+\delta\boldsymbol{x},\boldsymbol{y}^{(0)}+\delta\boldsymbol{y})$ is legal and minimizes both TNS and WNS, while also reducing routed wirelength overhead.

Previous methods can be mainly categorized into path-based or net-based. Path-based approaches [6, 10, 14–22] apply heuristics to reshape critical paths. A representative technique is path straightening, where cells on critical paths are relocated to reduce delay. Recent works like OWARU [10] and Rsyn [5, 6] enhance this with legality awareness and quadratic placement. The core limitation of this category is its reliance on local structural rules rather than direct optimization of global timing metrics. Net-based approaches [7–9, 13, 23–25], on the other hand, frame the problem as a constrained search over discrete candidate locations. The Lagrangian Relaxation (LR) framework [7] is a prominent example that integrates timing into the objective to guide a local search. However, these methods are fundamentally limited by their reliance on heuristic scoring and the high computational cost of repeatedly evaluating numerous candidate locations, resulting in significant runtime overhead.

2.2 Timing Sensitivity Analysis: Forward vs. Backward

Timing sensitivity analysis, which assesses how cell movements affect timing, is a key aspect of timing-driven placement [13]. This analysis generally falls into two paradigms: forward and backward. **Forward sensitivity analysis** resembles forward-mode differentiation[1]. Generally speaking[2], given a function $\boldsymbol{y} = f(\boldsymbol{x})$, where $\boldsymbol{x} \in \mathcal{R}^n$ and $\boldsymbol{y} \in \mathcal{R}^m$, this paradigm computes the output change $d\boldsymbol{y}$ resulting from an input perturbation $d\boldsymbol{x}$ using the Jacobian matrix $J_f = \frac{\partial f}{\partial \boldsymbol{x}} \in \mathcal{R}^{m\times n}$. The relationship is expressed in differential form as $d\boldsymbol{y} = J_f(\boldsymbol{x})d\boldsymbol{x}$. Forward differentiation has a time complexity of $O(n^2m)$, making it efficient when $n \ll m$. Existing detailed placement methodologies [5–7] adopt this forward-mode paradigm, enumerating candidate cell movements and selecting the most beneficial ones through forward timing propagation (see Fig. 2). Although this approach is computationally efficient for single-cell moves, it lacks analytical guidance for cell movements and cannot optimize multiple cells in parallel. Furthermore, to accelerate evaluation, these methods typically perform local timing analysis by propagating timing information only a few hops, which prevents them from capturing a global view of the timing landscape [7, 13]. These limitations highlight the need for a fundamentally different approach that can provide analytical movement guidance and enable comprehensive, global timing optimization.

Conversely, **backward sensitivity analysis** resembles reverse-mode differentiation [26]. This method is particularly effective when the number of inputs vastly exceeds the number of outputs, like in neural network training (as shown in the top-right corner of Fig. 2), where we compute gradients of a single loss function with respect to millions of parameters. For a scalar output L, we apply the chain rule to compute the gradient $\nabla_x L$ with respect to the input vector $\boldsymbol{x}$. This gradient represents the sensitivity of the output to changes in the input. Using a first-order Taylor expansion, we can approximate the change in L for a small perturbation $\delta\boldsymbol{x}$ as:

$$\Delta L = (\nabla_x L)^\top \cdot \delta\boldsymbol{x} + \mathcal{O}(\|\delta\boldsymbol{x}\|^2)$$

This formulation aligns perfectly with TNS/WNS optimization, where the goal is to optimize a scalar timing metric with respect to numerous cell locations (see Fig. 2). By leveraging backward sensitivity analysis, our methodology fundamentally departs from conventional techniques, enabling direct, gradient-based optimization in this high-dimensional space.

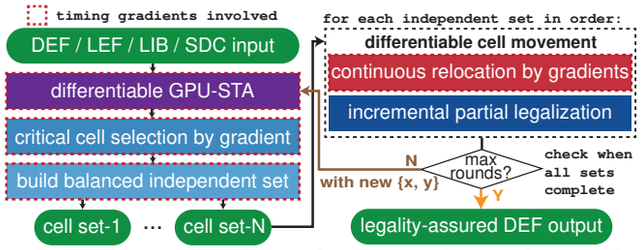

Figure 3: The overall workflow of our proposed LATTE framework.

3 The LATTE Framework

3.1 LATTE's Philosophy and Overview

LATTE employs backward-mode differentiation to approximate the change in timing metrics like TNS as a linear function of cell displacements:

$$\Delta\text{TNS} = \sum_{i\in V}\left(\frac{\partial\text{TNS}}{\partial x_i},\frac{\partial\text{TNS}}{\partial y_i}\right)\cdot\delta\boldsymbol{p}_i + \mathcal{O}(\|\delta\boldsymbol{p}_i\|^2). \quad (1)$$

where $\delta\boldsymbol{p}_i = (\delta x_i,\delta y_i)$. This linear approximation indicates that the optimal direction to move cell i to reduce TNS is along the negative gradient, $-\left(\frac{\partial\text{TNS}}{\partial x_i},\frac{\partial\text{TNS}}{\partial y_i}\right)$. Consequently, we propose the first non-rule-based and non-enumerative optimization direction for detailed placement.

Fig. 3 illustrates our differentiable timing-driven detailed placement framework, which iteratively refines an initial placement through a series of optimization rounds. Each round produces a legalized result and is structured around three core stages. First, our differentiable STA engine computes timing gradients, $\partial\text{TNS}/\partial\boldsymbol{p}_i$ and $\partial\text{WNS}/\partial\boldsymbol{p}_i$, to guide the optimization (Sec. 3.2). These gradients enable a precise, sensitivity-based selection of timing-critical cells (Sec. 3.3). To manage optimization dependencies, we partition these critical cells into multiple independent sets (Sec. 3.4). Second, a gradient-steered relocation module processes these independent sets as batches (Sec. 3.5). For each batch, this module formulates and solves a continuous optimization problem by relaxing alignment constraints to find optimal cell movements. Finally, an incremental partial legalization engine places the moved cells into valid

[1] For the discussion of forward-mode differentiation and backward-mode differentiation, please refer to [26].

[2] In Sec. 2.2, we mainly describe a general scenario for forward-mode differentiation and backward-mode differentiation. The equations in this section are presented as general mathematical expressions rather than being tied to any specific physical meaning.

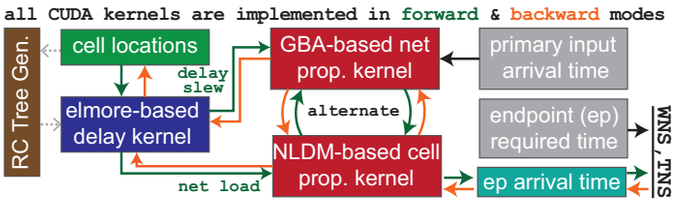

Figure 4: Computation graph of our differentiable STA engine, illustrating the forward and backward passes.

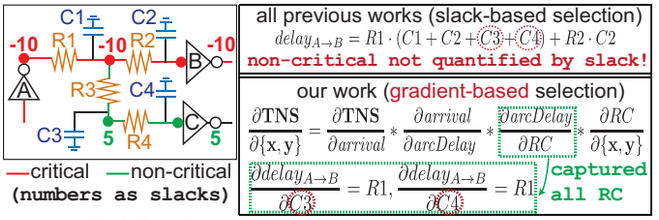

Figure 5: Slack-based versus gradient-based critical cell selection. The Elmore delay model reveals that cells on non-critical branches can influence the delay on the critical path. Therefore, reducing the load on these branches can reduce the critical path delay. Slack-based methods cannot quantify this influence, whereas gradient-based methods precisely measure and leverage this effect for optimization.

sites, ensuring the solution remains legal after each batch is processed (Sec. 3.6).

This work introduces a fundamentally new approach to timing-driven detailed placement. By leveraging differentiable optimization, our framework achieves theoretically optimal timing placement for individual cells under more general scenarios. Furthermore, we propose a novel independent set partitioning strategy that enables coherent optimization of multiple cells, leading to a more global and effective timing-driven placement solution (Sec. 3.7). The subsequent sections will detail each component of our proposed framework.

3.2 Our Differentiable STA Engine (Timing Kernels)

Inspired by INSTA [3], our differentiable STA engine decomposes timing analysis into two stages: delay calculation and timing propagation, as illustrated in Fig. 4. The first stage employs an Elmore-based delay kernel to compute closed-form, analytically differentiable expressions for RC tree delays, impulse responses, and capacitive loads. The second stage performs timing propagation by traversing the timing graph, alternating between two kernels: a Graph-Based Analysis (GBA) kernel for net delay propagation, and an Non-Linear Delay Model (NLDM)-based kernel for cell delay propagation [27], respectively. Unlike INSTA, which relies on external tools, our engine is a fully self-contained system. We integrate all computations into high-performance GPU kernels, treating cell locations as explicit optimization variables. This self-contained, GPU-native design significantly improves optimization precision, runtime efficiency, and stability.

3.3 Gradient-based Critical Cell Selection

We propose a gradient-based method to select critical cells for optimization, which overcomes key limitations of conventional slack-based approaches [5–7]. Gradients more accurately quantify the timing impact of variables like cell location and size. First, gradients can measure the timing impact of cells on non-critical branches that affect critical path delay, an effect slack-based methods cannot capture (Fig. 5)[3]. Second, as a path-based metric, slack assigns the same value to all cells on a critical path. This approach fails to distinguish between cells that fanout to different numbers of violating endpoints, a distinction that gradients reveal. Finally, gradients help identify the root cause of timing violations. If a path's delay is dominated by large intrinsic cell delays unaffected by relocation, the position gradients of its cells will be small. This correctly indicates that relocation is ineffective and that other optimizations strategies, such as gate sizing[4], are needed—a distinction that slack-based methods cannot make.

To implement this, our differentiable timing engine computes the TNS gradient w.r.t. the coordinates (x_i, y_i) of each cell i:

$$G_i = \left(\frac{\partial \text{TNS}}{\partial x_i}, \frac{\partial \text{TNS}}{\partial y_i}\right). \tag{2}$$

The gradient magnitude $\|G_i\|$ quantifies the end-to-end sensitivity of TNS to the location of cell i. A larger magnitude indicates a greater potential to improve TNS through relocation. In each optimization iteration, we rank all cells by their gradient magnitudes and select those with the highest values to form the critical cell set $\mathcal{C}_{critical}$. This gradient-based metric provides a precise, first-order estimate of the potential timing improvement from cell relocation. It inherently accounts for optimization paradigms like path strengthening [10] and load balancing [6] (see Fig. 5), thus eliminating the need for the manually-defined heuristic optimization paradigms used in traditional methods.

Algo. 1 Balanced Independent Set Partitioning

Input: Netlist $\mathcal{G} = (V,E)$, Critical cell set $\mathcal{C}_{critical}$, timing gradients $\boldsymbol{G}$, max balancing iterations T.
Output: A partition of $\mathcal{C}_{critical}$ into independent sets $\mathcal{I} = \{S_1,\ldots,S_K\}$.

```
1: // Phase 1: Greedy Coloring Algorithm
2: Sort C_critical in descending order of gradient magnitude ||G_c||
3: I ← ∅
4: for c ∈ C_critical do
5:     Find the first set S ∈ I s.t. S∪{c} is an independent set.
6:     if such a set S exists then
7:         S ← S∪{c}
8:     else
9:         I ← I∪{{c}}                                  ▷ Create a new set
10: // Phase 2: Iterative Balancing
11: for t = 1 to T do
12:     S_max ← argmax_{S∈I} |S|; S_min ← argmin_{S∈I} |S|
13:     if |S_max| − |S_min| ≤ 1 then break
14:     Find c* = argmin_{c∈S_max | S_min∪{c} is an independent set} ||G_c||
15:     if no such c* exists then break                  ▷ No feasible move
16:     S_max ← S_max \ {c*}; S_min ← S_min ∪ {c*}
17: return I \ {∅}                                       ▷ Remove empty sets
```

3.4 Balanced Independent Set Partitioning

To enable parallel optimization, we partition the critical cell set $\mathcal{C}_{critical}$ into disjoint independent sets[5] $\{S_1,\ldots,S_K\}$, allowing all cells within each set S_i to be relocated concurrently. Our proposed algorithm, detailed in Algorithm 1, operates in two phases.

First, a greedy coloring algorithm creates the initial partition. To prioritize high-impact cells, it processes them in descending order of gradient magnitude ($\|G_i\|$), assigning each to the first compatible set or creating a new one (lines 2-9). Second, a balancing phase refines this partition to correct size imbalances inherent to the greedy strategy, ensuring a balanced workload for parallel processing. This phase iteratively transfers a cell from the largest set ($S_{\max}$) to the smallest ($S_{\min}$) (line 12). To minimize timing impact, the selected cell is legally movable and has the minimum gradient magnitude in its source set (lines 14). The balancing process terminates when set sizes are nearly equal ($|S_{\max}|-|S_{\min}| \le 1$) or a maximum number of iterations is reached. This two-phase strategy effectively prioritizes timing-critical cells while producing balanced sets for efficient parallel optimization.

3.5 Parallel Batch-based Differentiable Cell Movement

3.5.1 Continuous Relocation by Gradients. For each independent set S, we formulate cell relocation as a continuous multi-objective optimization problem (Fig. 6). We use FLUTE [29] to initialize Steiner points and

[3] For example, simply looking at the slack values on the left part of Fig. 5 cannot reveal by how much reducing the load on non-critical branches (e.g., C3 and C4) by one unit will decrease the delay along the critical path $delay_{A\to B}$.

[4] Post placement optimization encompasses multiple techniques, including detailed placement, gate sizing, and buffer insertion. In this work, we focus on detailed placement. Nevertheless, it is important to note that our backward sensitivity analysis framework can reveal the underlying causes of timing degradation and can be extended to other optimization techniques.

[5] *Independent set* is a term in graph theory, please refer to [28] for more details.

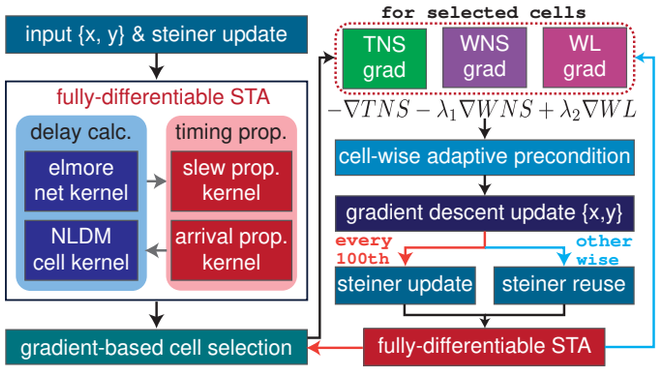

Figure 6: Parallel batch-based differentiable cell relocation on GPU. Each batch comprises an independent set of cells, enabling highly parallelized relocation operations on the GPU for efficient detailed placement.

periodically update their positions as cell coordinates change. In each inner optimization loop, we fix the positions of all cells outside S and the Steiner points to localize the optimization to the current batch.

We formulate the objective function for cells in S as a weighted sum of TNS, WNS, and wirelength:

$$\min_{\boldsymbol{x},\boldsymbol{y}} F(\boldsymbol{x},\boldsymbol{y}) = -\mathrm{TNS}(\boldsymbol{x},\boldsymbol{y}) - \lambda_1 \mathrm{WNS}(\boldsymbol{x},\boldsymbol{y}) + \lambda_2 \mathrm{WL}(\boldsymbol{x},\boldsymbol{y}), \tag{3}$$

where $\boldsymbol{x}$ and $\boldsymbol{y}$ are the coordinates of cells in S. Our differentiable formulation allows us to compute the exact gradient $\boldsymbol{g} \in \mathbb{R}^{|S|\times 2}$ of all objectives w.r.t. cell positions, providing a precise optimization direction.

Unlike previous methods that use indirect or heuristic weighting [5–7], our approach directly and simultaneously optimizes all critical metrics. This eliminates the need for manual tuning or ad hoc trade-offs. We use the Adam optimizer [30] with a Warmup + Cosine Annealing learning rate schedule to ensure robust convergence. The warmup phase promotes aggressive exploration in early iterations, while cosine annealing ensures smooth, stable convergence.

3.5.2 Cell-wise Adaptive Warm-up Preconditioner. Applying Adam directly to our objective can cause severe numerical instability or even divergence (Fig. 8). This challenge stems from the nature of STA, where long signal propagation paths cause timing-driven gradients to vary by orders of magnitude across cells. This phenomenon is analogous to exploding and vanishing gradients in deep neural networks and can cripple optimization if unaddressed.

To overcome this, we introduce the Cell-wise Adaptive Warm-up Preconditioner (CAWP), inspired by large-batch training techniques in deep learning [31]. While deep learning methods typically normalize gradients at the layer level, we adapt this idea to placement by normalizing on a per-cell basis. This granular, per-cell approach directly addresses the unique challenges of timing-driven placement.

Specifically, we construct a diagonal preconditioning matrix $\mathcal{P}$. Each diagonal entry p_i for cell i is based on the root mean square (RMS) of its gradient norm over an initial warm-up window of T_0 iterations:

$$p_i = \frac{\tau}{\sqrt{\frac{1}{T_0}\sum_{t=0}^{T_0-1} \|\boldsymbol{g}_i^{(t)}\|_2^2 + \epsilon}}, \quad \mathcal{P} = \mathrm{diag}(p_1,\ldots,p_{|S|}) \tag{4}$$

Here, $\boldsymbol{g}_i^{(t)}$ is the 2D gradient for cell i at iteration t, τ is a global scaling hyperparameter, and ϵ is a small constant for numerical stability[6]. We then pass the preconditioned gradient $\tilde{\boldsymbol{g}}^{(t)} = \mathcal{P}\cdot\boldsymbol{g}^{(t)}$ to the Adam optimizer. The idea of CAWP is to ensure stable and rapid convergence by first normalizing gradients to prevent initial instability and then allowing Adam's adaptive moment estimation to take over, a strategy whose efficacy is validated by our experiments (Fig. 8).

3.6 Incremental Partial Legalization and Bad-Move Filtering

After differentiable relocation, we must legalize the continuous cell locations onto the discrete placement grid. We devise a **incremental partial legalization** scheme that prioritizes timing-critical cells. When processing an independent set S_i, we treat previously legalized sets $(S_1,\ldots,S_{i-1})$ and non-critical cells $(\overline{S} = V \setminus \bigcup_{j=1}^{K} S_j)$ as fixed obstacles. We ignore subsequent sets $(S_{i+1},\ldots,S_K)$ during this step. We use the Tetris [33] and Abacus [34] legalization algorithms to resolve overlaps within each batch. This incremental approach complements our partitioning strategy (Algo. 1), which places the most critical cells in earlier sets. This gives them precedence in securing valid, timing-aware placements and protects optimization gains.

Legalization is a purely geometric operation, agnostic to timing, which risks nullifying our optimization gains. To mitigate this, we introduce a **bad-move filtering** safeguard. After each round, we perform a full timing analysis and if the timing is degraded, we will reject all the moves in this round. This guarantees *monotonic timing improvement* throughout the iterative refinement process.

[6] When the iteration number t is less than T_0, we use the gradients from the first t iterations to compute the preconditioning matrix in a similar way like Eq. (4).

Table 1: Performance comparison with state-of-the-art detailed placers on the TILOS [12] and ISPD 2013 [32] benchmarks, starting from timing- and routability-optimized placements generated by a leading commercial global placer.

Design (7nm) (#cells)	Method	WL (μm)	WNS ($10^3 ps$)	TNS ($10^5 ps$)
NVDLA (173K cells) (128 macros)	Initial	2030820	-1.79	-119.33
	DP4.0T	2082534	-1.85	-123.89
	Ours	2091572	-1.73	-104.29 (-12.6%)
ARIANE133 (117K cells) (133 macros)	Initial	926494	-1.91	-121.97
	DP4.0T	962163	-2.06	-126.89
	Ours	967562	-1.68	-95.02 (-22.1%)
ARIANE136 (113K cells) (136 macros)	Initial	962224	-1.33	-117.52
	DP4.0T	1002536	-4.23	-121.30
	Ours	1009730	-1.14	-101.86 (-13.3%)
MEMPOOL (162K cells) (20 macros)	Initial	1074585	-8.67	-692.96
	DP4.0T	1100853	-11.00	-851.98
	Ours	1162561	-8.38	-643.03 (-7.2%)
netcard_fast (319K cells)	Initial	3740027	-5.52	-781.06
	Rsyn	4256914	-4.68	-699.70
	DP4.0T	3820897	-5.60	-794.78
	Ours	3885548	-4.89	-614.50 (-21.3%)
edit_dist_fast (70K cells)	Initial	600363	-2.20	-48.61
	Rsyn	664636	-2.33	-48.25
	DP4.0T	614247	-2.42	-50.41
	Ours	626087	-2.05	-43.86 (-9.8%)

3.7 Discussion: Backward-Mode Optimization Paradigm

Our framework emulates backward-mode differentiation. We back-propagate gradients from global timing metrics to cell locations, deriving a mathematically-grounded optimization direction for all cells simultaneously. This backward-mode paradigm yields two key advantages. **Superior Optimality for General Cells.** Prior methods find optimal locations for simple single-input, single-output (SISO) cells. However, they resort to heuristics for more complex multi-input, single-output (MISO) cells [5, 6]. In contrast, our gradient-based approach precisely computes the optimal timing-aware position for general MISO cells. **Holistic Batch Optimization.** Our framework optimizes all cells in an independent set in parallel, simultaneously considering their positional interdependencies. This holistic optimization is intractable for forward-mode methods, which face a combinatorial explosion when evaluating multi-cell move candidates.

4 Experimental Results

We evaluate our proposed algorithm against several state-of-the-art timing-driven detailed placers. Our benchmark suite includes 15 diverse designs from TILOS [12], ISPD 2013 [32], and IWLS [11], ranging from standard-cell-only to macro-heavy. For a fair comparison, we generate initial placements using a variety of industry and academic global placers [35, 36]. We measure all reported metrics after an early global route

Table 2: Comprehensive comparison with state-of-the-art detailed placers on IWLS [11] benchmarks, starting from DREAMPlace [35] initial placements in ASAP 7nm technology. RT denotes the runtime.

Design (7nm)	Initial Placement			Rsyn [5, 6]				DP4.0T [7]				Ours			
	WL (μm)	WNS ($10^3 ps$)	TNS ($10^5 ps$)	WL (μm)	WNS ($10^3 ps$)	TNS ($10^5 ps$)	RT (s)	WL (μm)	WNS ($10^3 ps$)	TNS ($10^5 ps$)	RT (s)	WL (μm)	WNS ($10^3 ps$)	TNS ($10^5 ps$)	RT (s)
ac97_top	31042	-19.98	-117.14	82457	-19.37	-111.25	370	64096	-31.54	-149.13	944	32447	-17.38	-99.25	433
aes	28909	-7.30	-24.96	41457	-6.79	-20.39	59	29840	-7.37	-24.26	980	29411	-6.78	-20.54	689
des	7195	-1.04	-1.00	9149	-0.88	-0.82	4	7508	-1.06	-0.95	362	7704	-0.89	-0.83	723
des3	13394	-12.14	-5.73	19518	-10.09	-5.82	4	14873	-14.84	-6.00	641	13894	-10.50	-5.65	534
fpu	68222	-48.00	-49.37	100765	-43.88	-42.59	93	71085	-49.16	-49.55	1350	69938	-44.06	-45.11	8221
i2c_master_top	2478	-21.99	-10.20	4308	-21.96	-12.13	3	7675	-52.47	-21.05	296	2579	-17.83	-10.07	123
mc_top	24490	-34.84	-139.71	32039	-37.77	-135.88	3	29177	-35.94	-144.65	843	25455	-32.58	-120.46	344
pci_bridge32	63249	-24.84	-256.78	97292	-33.79	-207.93	14	fail	fail	fail	fail	65503	-21.06	-214.31	714
tv80s	16801	-15.02	-40.42	22541	-13.43	-34.59	2	17321	-17.85	-46.60	907	17814	-12.86	-34.60	1382
Geo Mean.	1.000	1.000	1.000	1.541	0.970	0.914	0.018	1.330	1.249	1.149	1.021	1.041	0.876	0.880	1.000

Table 3: Comprehensive comparison with state-of-the-art detailed placers on IWLS [11] benchmarks, starting from DREAMPlace 4.0 [36] initial placements in ASAP 7nm technology. RT denotes the runtime.

Design (7nm)	Initial Placement			Rsyn [5, 6]				DP4.0T [7]				Ours			
	WL (μm)	WNS ($10^3 ps$)	TNS ($10^5 ps$)	WL (μm)	WNS ($10^3 ps$)	TNS ($10^5 ps$)	RT (s)	WL (μm)	WNS ($10^3 ps$)	TNS ($10^5 ps$)	RT (s)	WL (μm)	WNS ($10^3 ps$)	TNS ($10^5 ps$)	RT (s)
ac97_top	31290	-18.68	-115.46	52253	-20.39	-102.71	159	31606	-19.61	-114.03	857	32597	-17.03	-94.32	296
aes	29178	-7.42	-24.90	46895	-7.53	-18.50	35	29383	-7.69	-26.30	1031	29672	-7.89	-22.15	502
des	7419	-1.05	-0.99	9757	-0.89	-0.82	3	7514	-1.10	-1.05	374	7754	-0.89	-0.83	841
des3	13700	-13.34	-6.54	20358	-10.00	-6.07	4	13797	-12.73	-6.04	528	14141	-10.52	-5.20	397
fpu	68521	-46.75	-47.73	108935	-48.26	-47.68	75	69479	-46.50	-47.33	1343	70273	-45.07	-45.43	8726
i2c_master_top	2507	-17.73	-10.12	4108	-28.90	-13.39	2	2530	-19.16	-10.19	290	2542	-19.33	-10.48	90
mc_top	25125	-44.84	-141.09	32660	-40.30	-146.28	3	25324	-38.70	-139.65	907	26103	-44.88	-150.80	743
pci_bridge32	fail	fail	fail	-	-	-	-	-	-	-	-	-	-	-	-
tv80s	17176	-15.40	-46.11	23871	-14.21	-35.11	1	17369	-14.89	-43.14	847	18209	-13.45	-40.22	1332
Geo Mean.	1.000	1.000	1.000	1.469	0.985	0.929	0.008	1.010	0.990	0.993	0.897	1.034	0.938	0.903	1.000

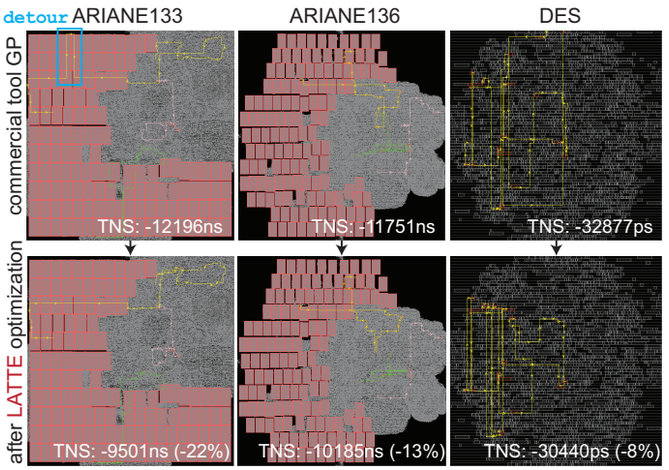

Figure 7: Visualization of exactly same top critical paths before and after LATTE DP optimization. Detours are optimized by timing gradients.

with a leading commercial tool to validate end-of-flow quality. We run all experiments on a Linux platform with an NVIDIA A100 GPU (96GB), an AMD EPYC 7742 CPU (64 cores), and 1.5TB of RAM.

4.1 Head-to-Head Comparison with State-of-the-Art Detailed Placers

We perform direct comparisons with the Lagrangian-based timing-driven detailed placer DREAMPlace 4.0 TCAD (DP4.0T) [7] and the path-strengthening-based placer Rsyn [5, 6]. To rigorously assess robustness, we utilize three distinct sets of initial placements generated in the ASAP 7nm node [37]: (1) a timing- and routability-aware commercial global placer, (2) the wirelength-driven academic placer DREAMPlace [35], and (3) the timing-driven academic placer DREAMPlace 4.0 [36]. This diverse set of starting points enables a thorough assessment of each placer's optimization capabilities.

4.1.1 Optimization from a Commercial Placer Baseline. Table 1[7] compares routed wirelength and timing on initial placements from a leading commercial tool [8]. These input placement results are already highly optimized, making further improvements challenging. Nonetheless, our method consistently improves timing, achieving superior TNS across all test cases. On average, our method improves TNS by 14.6% and WNS by 8.5%, with only a 4.8% increase in routed wirelength. These results show that even well-tuned commercial input placement results have significant room for optimization. Compared to other academic placers, our approach improves TNS by 24.5% over DP4.0T and 11.9% over Rsyn, while offering a more efficient wirelength trade-off than Rsyn's 12.3% increase. This demonstrates that our gradient-based optimization effectively balances timing and wirelength to find superior solutions.

4.1.2 Optimization from Academic Global Placers Baseline. Table 2 and Table 3 show results on initial placements from the wirelength-driven DREAMPlace [35] and timing-aware DREAMPlace 4.0 [36]. This analysis reveals limitations in competing detailed placers. While Rsyn is fast, its greedy heuristics aggressively optimize timing at the cost of a 54.1% wirelength increase, which results in significant dynamic power increase. In contrast, DP4.0T relies on local timing propagation, which lacks a global view and leads to suboptimal decisions and degraded timing. LATTE improves TNS by 29.7% and wirelength by 44.4% on average over these placers, with only a minor wirelength increase from the baseline. Our gradient-based approach achieves a superior timing-wirelength trade-off, delivering better timing with runtimes comparable to DP4.0T and minimal wirelength overhead. This balance is crucial for design

[7] Rsyn fails on designs with macros (primarily SRAMs) due to liberty parsing errors. [8] We can not disclose the commercial tools names due to licensing agreements.

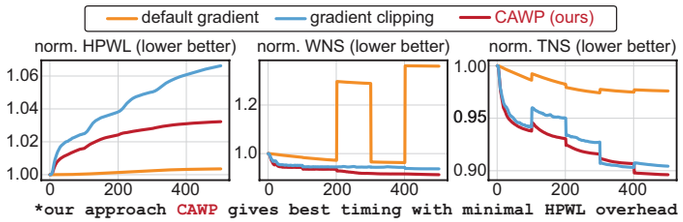

Figure 8: Evolution of normalized HPWL, WNS, and TNS for three gradient manipulation methods across DP iterations. All metrics are normalized by initial values at iteration 0. Steiner trees are updated every 100th iteration.

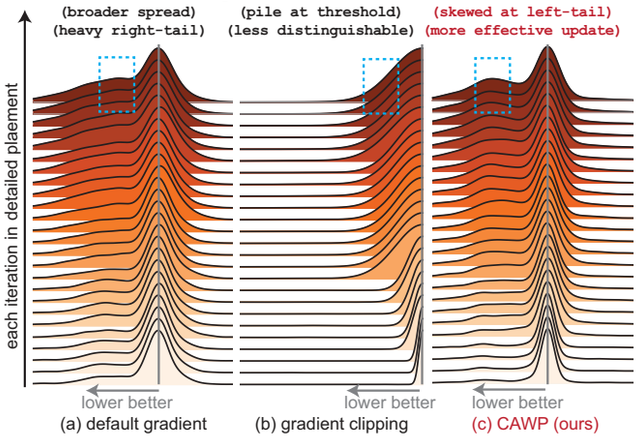

Figure 9: Gradient magnitude histograms for three manipulation methods. The horizontal axis shows the log-scale gradient norm, and the vertical axis shows the iteration index. Each row depicts the gradient norm distribution per iteration.

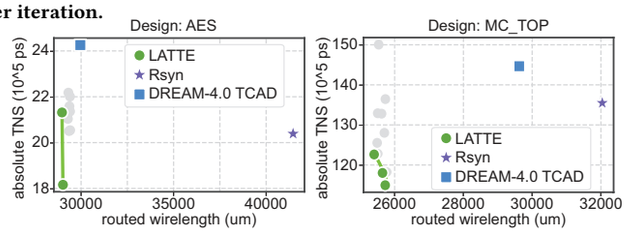

Figure 10: `LATTE`'s tunable timing–WL settings achieve the best Pareto frontier on `aes` and `mc_top` compared with `Rsyn` and `DP4.0T` that are non-tunable and yield one point each.

closure, as production flows require co-optimization of timing and wirelength. Combined with the results from the commercial baseline, these experiments show that LATTE consistently improves timing regardless of the initial placement's quality or optimization focus.

4.2 Ablation Study: Gradient Manipulation Methods

We further evaluate three gradient manipulation methods on the ac97_top design: 1) default gradient, 2) gradient value clipping, and 3) our proposed CAWP (Sec. 3.5.2). As shown in Fig. 8, the default gradient method barely improves TNS and even diverges on WNS. Default gradients apply excessively large updates to cells on critical paths, causing divergence and preventing convergence to an optimal solution. Compared to gradient clipping, CAWP achieves lower Half-Perimeter Wirelength (HPWL) and exhibits smaller fluctuations in both HPWL and TNS across Steiner-point updates, indicating greater stability. Fig. 9 corroborates these findings, showing that the gradient distribution for CAWP exhibits a significantly more pronounced leftward skew, accumulating more mass in low-to-moderate gradient range over iterations compared with the other two method. This indicates that our method more effectively moves cells toward better optima.

4.3 Case Study: Wirelength-Timing Pareto Frontier

Fig. 10 illustrates LATTE's ability to explore the wirelength-timing tradeoff. By adjusting the wirelength weight in Eq. (3), LATTE generates a Pareto frontier of solutions, offering designers a spectrum of choices instead of a single point. These illustrations demonstrate that most LATTE solutions are Pareto-superior to those from Rsyn and DP4.0T, achieving better wirelength and timing simultaneously. On the aes benchmark, LATTE reduces wirelength by nearly 30% relative to Rsyn while also improving timing. At a similar wirelength, LATTE improves TNS by about 25% compared to DP4.0T. This capability provides designers with a richer set of high-quality options to meet diverse design requirements.

Table 4: Ablation study on integrating wirelength-driven and timing-driven detailed placers on IWLS [11] benchmark designs in a 7*nm* technology node.

Design (7*nm*)	ABCD [1]		LATTE + ABCD		LATTE	
	WL (*um*)	TNS ($10^5 ps$)	WL (*um*)	TNS ($10^5 ps$)	WL (*um*)	TNS ($10^5 ps$)
ac97_top	30821	-110.77	30832	-105.82	32597	-94.32
aes	28908	-25.07	28671	-24.69	29672	-22.15
des	7250	-0.97	7207	-0.98	7754	-0.83
des3	13393	-6.35	13395	-5.47	14141	-5.20
fpu	67835	-47.73	67054	-48.67	70273	-45.43
i2c_master_top	2467	-10.31	2426	-11.20	2542	-10.48
mc_top	24552	-149.89	24342	-130.03	26103	-150.80
tv80s	16849	-44.33	16702	-42.53	18209	-40.22
Geo. Mean	0.950	1.101	0.943	1.062	1.000	1.000

4.4 Ablation Study: Integrating Wirelength-Driven and Timing-Driven Detailed Placers

We conduct an ablation study with the wirelength-driven placer ABCDPlace (ABCD) [1] to determine if it can repair wirelength degradation after timing optimization. We evaluate three scenarios: (1) ABCD alone, (2) LATTE alone, and (3) LATTE followed by ABCD.

The results in Table 4 reveal several key insights. Comparing LATTE with LATTE + ABCD shows that ABCD effectively repairs wirelength but degrades timing by 6.2%. This indicates that subsequent wirelength optimization can harm previously improved timing. Furthermore, comparing ABCD with LATTE + ABCD, both methods achieve similar routed wirelength, but LATTE + ABCD delivers superior TNS. These strongly encouraging findings indicate that, although existing heuristic-based detailed placers [5–7] often trade timing for wirelength as shown in Fig. 10, our LATTE framework achieves a fundamental improvement in the timing landscape that persists through subsequent wirelength refinement.

5 Conclusion

In this work, we introduced LATTE, the first legality-assured, end-to-end differentiable framework for timing-driven detailed placement. By leveraging exact, back-propagated gradients from global timing metrics, LATTE overcomes the fundamental limitations of traditional heuristic and enumeration-based methods. Our approach synergizes continuous, gradient-steered optimization of multiple cells with discrete incremental legalization, enabling holistic performance improvements while rigorously maintaining placement validity. Comprehensive experiments on 15 industrial 7*nm* designs demonstrate that LATTE consistently achieves superior timing, improving TNS by up to 20% over state-of-the-art academic placers with minimal wirelength impact. By replacing local heuristics with a mathematically-grounded, global optimization approach, LATTE represents a significant step forward in detailed placement, offering a powerful and robust solution for timing closure in modern physical design.

Acknowledgments

We gratefully acknowledge the anonymous reviewers for their valuable feedback. We also thank Yanqing Zhang, Wen-Hao Liu, Zhili Xiong, and Yunsheng Bai at NVIDIA for their insightful feedback on this work.

References

[1] Y. Lin, W. Li, J. Gu, H. Ren, B. Khailany, and D. Z. Pan, "ABCDPlace: Accelerated Batch-based Concurrent Detailed Placement on Multi-threaded CPUs and GPUs," IEEE transactions on computer-aided design of integrated circuits and systems, vol. 39, no. 12, pp. 5083–5096, 2020.

[2] 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," in 2025 44th ACM/IEEE International Conference on Computer-Aided Design (ICCAD), 2025.

[3] 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.

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

[5] M. Fogaça, G. Flach, J. Monteiro, M. Johann, and R. Reis, "Quadratic timing objectives for incremental timing-driven placement optimization," in 2016 IEEE International Conference on Electronics, Circuits and Systems (ICECS), pp. 620–623, IEEE, 2016.

[6] G. Flach, M. Fogaça, J. Monteiro, M. Johann, and R. Reis, "Drive strength aware cell movement techniques for timing driven placement," in Proceedings of the 2016 on International Symposium on Physical Design, pp. 73–80, 2016.

[7] 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.

[8] G. Flach, J. Monteiro, M. Fogaça, J. Puget, P. Butzen, M. Johann, and R. Reis, "An incremental timing-driven flow using quadratic formulation for detailed placement," in 2015 IFIP/IEEE International Conference on Very Large Scale Integration (VLSI-SoC), pp. 1–6, IEEE, 2015.

[9] C.-C. Huang, Y.-C. Liu, Y.-S. Lu, Y.-C. Kuo, Y.-W. Chang, and S.-Y. Kuo, "Timing-driven cell placement optimization for early slack histogram compression," in Proceedings of the 53rd Annual Design Automation Conference, pp. 1–6, 2016.

[10] J. Jung, G.-J. Nam, L. N. Reddy, I. H.-R. Jiang, and Y. Shin, "OWARU: Free space-aware timing-driven incremental placement with critical path smoothing," IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems, vol. 37, no. 9, pp. 1825–1838, 2017.

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

[12] 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.

[13] D. Mangiras, A. Stefanidis, I. Seitanidis, C. Nicopoulos, and G. Dimitrakopoulos, "Timing-driven placement optimization facilitated by timing-compatibility flip-flop clustering," IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems, vol. 39, no. 10, pp. 2835–2848, 2019.

[14] A. Chowdhary, K. Rajagopal, S. Venkatesan, T. Cao, V. Tiourin, Y. Parasuram, and B. Halpin, "How accurately can we model timing in a placement engine?," in Proceedings of the 42nd annual Design Automation Conference, pp. 801–806, 2005.

[15] A. Bock, S. Held, N. Kämmerling, and U. Schorr, "Local search algorithms for timing-driven placement under arbitrary delay models," in Proceedings of the 52nd Annual Design Automation Conference, pp. 1–6, 2015.

[16] S. Kim, S. Do, and S. Kang, "Fast predictive useful skew methodology for timing-driven placement optimization," in 2017 54th ACM/EDAC/IEEE Design Automation Conference (DAC), pp. 1–6, IEEE, 2017.

[17] V. Livramento, R. Netto, C. Guth, J. L. Güntzel, and L. C. D. Santos, "Clock-tree-aware incremental timing-driven placement," ACM Transactions on Design Automation of Electronic Systems (TODAES), vol. 21, no. 3, pp. 1–27, 2016.

[18] H. Ren, D. Z. Pan, C. J. Alpert, G.-J. Nam, and P. Villarrubia, "Hippocrates: First-do-no-harm detailed placement," in 2007 Asia and South Pacific Design Automation Conference, pp. 141–146, IEEE, 2007.

[19] T. Luo, D. A. Papa, Z. Li, C. N. Sze, C. J. Alpert, and D. Z. Pan, "Pyramids: an efficient computational geometry-based approach for timing-driven placement," in 2008 IEEE/ACM International Conference on Computer-Aided Design, pp. 204–211, IEEE, 2008.

[20] M. D. Moffitt, D. A. Papa, Z. Li, and C. J. Alpert, "Path smoothing via discrete optimization," in Proceedings of the 45th annual Design Automation Conference, pp. 724–727, 2008.

[21] N. Viswanathan, G.-J. Nam, J. A. Roy, Z. Li, C. J. Alpert, S. Ramji, and C. Chu, "ITOP: Integrating timing optimization within placement," in Proceedings of the 19th international symposium on Physical design, pp. 83–90, 2010.

[22] T.-C. Lee and Y.-L. Li, "Incremental timing-driven placement with approximated signoff wire delay and regression-based cell delay," IEEE Transactions on Very Large Scale Integration (VLSI) Systems, vol. 27, no. 10, pp. 2434–2446, 2019.

[23] C. Guth, V. Livramento, R. Netto, R. Fonseca, J. L. Güntzel, and L. Santos, "Timing-driven placement based on dynamic net-weighting for efficient slack histogram compression," in Proceedings of the 2015 Symposium on International Symposium on Physical Design, pp. 141–148, 2015.

[24] G. Wu and C. Chu, "Two approaches for timing-driven placement by Lagrangian relaxation," IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems, vol. 36, no. 12, pp. 2093–2105, 2017.

[25] V. Livramento, C. Guth, R. Netto, J. L. Güntzel, and L. C. dos Santos, "Exploiting non-critical Steiner tree branches for post-placement timing optimization," in 2015 IEEE/ACM International Conference on Computer-Aided Design (ICCAD), pp. 528–535, IEEE, 2015.

[26] B. van den Berg, T. Schrijvers, J. McKinna, and A. Vandenbroucke, "Forward-or reverse-mode automatic differentiation: What's the difference?," Science of Computer Programming, vol. 231, p. 103010, 2024.

[27] "PrimeTime User Guide: Advanced timing analysis," V-2023.03, Synopsys online Documentation, 2023.

[28] "Independent set (graph theory)." https://en.wikipedia.org/wiki/Independent_set_(graph_theory). Accessed: 2025-11-13.

[29] 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, vol. 27, no. 1, pp. 70–83, 2007.

[30] D. P. Kingma and J. Ba, "Adam: A method for stochastic optimization," arXiv preprint arXiv:1412.6980, 2014.

[31] Y. You, I. Gitman, and B. Ginsburg, "Large batch training of convolutional networks," arXiv preprint arXiv:1708.03888, 2017.

[32] M. M. Ozdal, C. Amin, A. Ayupov, S. M. Burns, G. R. Wilke, and C. Zhuo, "An improved benchmark suite for the ISPD-2013 discrete cell sizing contest," in Proceedings of the 2013 ACM International symposium on Physical Design, pp. 168–170, 2013.

[33] T.-C. Chen, Z.-W. Jiang, T.-C. Hsu, H.-C. Chen, and Y.-W. Chang, "NTUplace3: An analytical placer for large-scale mixed-size designs with preplaced blocks and density constraints," IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems, vol. 27, no. 7, pp. 1228–1240, 2008.

[34] P. Spindler, U. Schlichtmann, and F. M. Johannes, "Abacus: Fast legalization of standard cell circuits with minimal movement," in Proceedings of the 2008 international symposium on Physical design, pp. 47–53, 2008.

[35] Y. Lin, Z. Jiang, J. Gu, W. Li, S. Dhar, H. Ren, B. Khailany, and D. Z. Pan, "Dreamplace: Deep learning toolkit-enabled gpu acceleration for modern vlsi placement," IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems, vol. 40, no. 4, pp. 748–761, 2020.

[36] P. Liao, S. Liu, Z. Chen, W. Lv, Y. Lin, and B. Yu, "DREAMPlace 4.0: Timing-driven global placement with momentum-based net weighting," in 2022 Design, Automation & Test in Europe Conference & Exhibition (DATE), pp. 939–944, IEEE, 2022.

[37] 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.