Disentangled Differentiable Timing-Power Co-Optimization with Quad-Gradient Gate Sizing

Zixuan Pan[2*], Zizheng Guo[1,3*], Yufan Du[5], Runsheng Wang[1,3,4], Yibo Lin[1,3,4†]
[1]School of Integrated Circuits, Peking University [2]School of EECS, Peking University [3]Institute of EDA, Peking University
[4]Beijing Advanced Innovation Center for Integrated Circuits [5]University of California, Los Angeles
{2300012747,gzz,r.wang,yibolin}@pku.edu.cn, nbsdyf@hotmail.com

ABSTRACT

Co-optimizing timing and power in modern VLSI designs remains challenging under realistic static timing analysis and standard-cell libraries. Classical gate sizing often scales poorly, while learning-based sizers behave as expensive black boxes with limited generality. Recent differentiable physical optimization enables gradient-based design flows, but existing approaches still struggle to stay aligned with library-based implementations and to provide controlled timing–power trade-offs. We propose a library-native quad-gradient gate sizing framework that leverages differentiable timing to derive structured guidance for timing and power, enabling more systematic and interpretable co-optimization in the standard-cell sizing space. On the ICCAD 2025 contest benchmarks, our framework achieves, on average, 40.4% larger reduction in TNS and 16.2% better total power change than the 1st-place contest flow.

1 INTRODUCTION

Gate sizing is a fundamental technique for timing closure and power optimization in standard-cell designs, and is NP-hard [29]. In practice, it must operate in a discrete standard-cell library with limited drive strengths and threshold voltages (Vth), under highly non-convex, nonlinear static timing analysis (STA). An ideal gate sizing optimizer should therefore (i) work directly in the discrete drive×Vth library space, (ii) provide physically meaningful and interpretable update directions, (iii) stay consistent with nonlinear STA, and (iv) scale to large designs.

Early work used continuous convex formulations, most notably geometric programming (GP), for gate sizing [11, 13, 16]. Such relaxations were later found to mismatch discrete standard-cell libraries, which motivated Lagrangian Relaxation (LR), sensitivity-based heuristics, and dynamic programming that operate more directly on discrete gate choices [2, 6, 7, 14, 15, 30, 31]. These methods are physically interpretable but typically run as CPU-based iterative STA with heuristic control, leading to high runtime and limited scalability, although parallelization on CPUs [20, 26, 33] can alleviate runtime.

More recently, machine learning (ML)–based methods, including RL and GNN/Transformer models [3, 22, 24, 27, 37], perform gate sizing directly in the discrete library and are typically implemented with GPU acceleration. However, they largely behave as black-box models with limited interpretability, while both their substantial training and their tendency toward design-specific overfitting limit their applicability across designs.

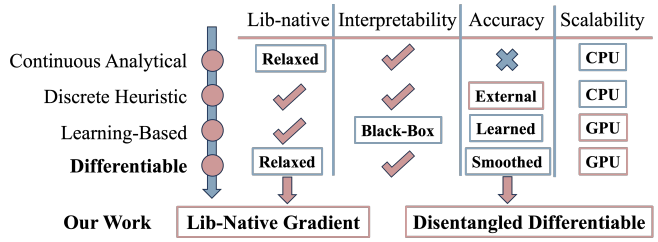

Figure 1: Conceptual comparison of gate sizing approaches along four axes: (i) discreteness w.r.t. the standard-cell library, (ii) interpretability of optimization, (iii) consistency with nonlinear STA, and (iv) scalability.

In the spirit of smoothed analytical placement [19, 21] and differentiable STA [12, 23], recent work on differentiable physical optimization expresses STA as a differentiable computation graph and runs gradient-based updates on GPUs for placement and sizing [8–10, 32, 35], making end-to-end optimization conceptually very appealing. However, existing differentiable gate sizing frameworks have three limitations: (i) they relax the discrete library to a continuous space and then project back, so gradient directions are not directly tied to legal moves in the discrete standard-cell library; (ii) the smoothing techniques used to make timing differentiable blur truly critical timing regions and spread attention into non-critical areas, weakening the focus where slack is tight; and (iii) timing improvement and power recovery are usually driven by a single combined loss or gradient, so emphasizing timing around critical paths can misjudge near-critical or timing-safe regions and unnecessarily hurt power.

Differentiable timing models are also approximate and can deviate from actual STA. Prior work on calibrating fast versus detailed timers has shown that lightweight corrections can significantly improve timing correlation [4, 5, 17, 18, 23, 34, 36], suggesting that bringing similar calibration into a differentiable sizing loop—still largely unexplored—can better align gradient with the timing behavior of STA engines.

In this work, we design a disentangled differentiable gate sizing framework with four goals: operating directly in the discrete drive×Vth library, keeping optimization interpretable, staying close to nonlinear STA, and scaling efficiently on GPUs. Figure 1 sketches how classical, ML-based, and differentiable approaches relate to these four goals and where our method lies. To achieve these goals, we make the following contributions:

- **Quad-Gradient library-native discrete gate sizing.** We optimize directly in a two-dimensional discrete library space indexed by drive strength and Vth, and compute directional subgradients aligned with library moves, ensuring consistency between gradients and discrete cell replacement.
- **A disentangled differentiable framework with decoupled timing–power gradients.** On top of a shared forward timing evaluator, we use two backward branches: one focuses on timing updates near critical paths to mitigate criticality diffusion,

*Equal contribution. †Corresponding author. This project is supported in part by the Natural Science Foundation of Beijing, China (Grant No. Z230002) and the National Science and Technology Major Project (2021ZD0114702).

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

ACM ISBN 979-8-4007-2254-7/2026/07.
https://doi.org/10.1145/3770743.3804047

and the other targets timing-safe regions for conservative, stable power recovery, decoupling conflicting objectives at the gradient level.

- **Multi-level physical calibration from reference STA.** We introduce calibration mechanisms within the differentiable framework so that timing quantities align in magnitude and trend with reference STA, ensuring consistent physical information in the computation graph.
- **Strong empirical gains on ICCAD 2025 Contest benchmarks.** On the ICCAD 2025 contest benchmarks [25], our framework achieves, on average, 40.4%*pt* larger TNS reduction and 16.2%*pt* better total power change than the best available contest flow.

The remainder of this paper is organized as follows: Sec. 2 reviews STA basics, prior differentiable timing, and our 2D sizing formulation, Sec. 3 presents the disentangled differentiable framework, Sec. 4 reports experimental results, and Sec. 5 concludes.

2 PRELIMINARIES

This section reviews the basics of timing and power objectives, the structure of a differentiable STA engine, and the timing-power co-optimization formulation for gate sizing.

2.1 Timing and Power Objectives

STA models the circuit as a directed acyclic graph (DAG) and computes signal transition arrival times in order to determine circuit performance. In this DAG, vertices represent pins, each associated with timing properties like arrival time $AT(\cdot)$, required arrival time $RAT(\cdot)$, and $slew(\cdot)$. Directed edges represent the timing arcs that propagate delay and slew between pins through cells and nets [1]. For the cell arcs, the nonlinear delay model (NLDM) defines output delay and slew using two look-up tables (LUTs) for each arc type. These LUTs are queried with input slew and output capacitive load using bilinear interpolation. For the net arcs, the Elmore delay model based on RC trees is widely used in early-stage optimization because it is simple and differentiable, and is usually calibrated to improve accuracy [18].

In VLSI gate sizing task, the sizes of logic gates (i.e. cells) determine the specific LUTs and pin capacitances used during timing propagation and thus have a profound impact on circuit timing. For every group of logically-equivalent gate types (e.g., AND2), the standard cell library provides many different physical implementations ("sizes" in our paper) with different drive strengths (e.g., ×1, ×4) combined with different threshold voltages (e.g. LVT, RVT). For a gate g with chosen size s_g (e.g., AND2x1_LVT), its delay and slew are defined as follows,

$$d_{u\to v}^{\text{cell } g} = LUT_{delay}^{u\to v}\big(slew(u),\ load(v);\ s_g\big),$$

$$slew(v) = LUT_{slew}^{u\to v}\big(slew(u),\ load(v);\ s_g\big).$$

Given the cell and net delay models, the arrival times and slews are propagated through the circuit topologically level by level. During the traversal, maximum arrival times from fanin pins u to every pin v are selected in $AT(v)$ to cover worst-case scenarios:

$$AT(v) = \max_{(u\to v)\in E} \{\, AT(u) + d_{u\to v} \,\}.$$

During physical optimization, we normally focus on setup-time analysis and optimization (i.e. max scenario) following the formulation in [25]. Each timing endpoint p (a register D pin or an output port) yields a slack value by subtracting its required arrival time $RAT(p)$ with its arrival time:

$$Slack(p) = RAT(p) - AT(p),$$

The final timing targets are then computed by accumulating these slacks across all endpoints $\mathcal{E}$, yielding worst negative slack (WNS) and total negative slack (TNS):

$$WNS = \min_{p\in\mathcal{E}} Slack(p), \qquad TNS = \sum_{p\in\mathcal{E}} \min\{0, Slack(p)\}.$$

For the power objective, we use total power consisting of leakage, internal, and switching power [25] using nonlinear power model (NLPM) similar to NLDM model for timing.

2.2 Differentiable Timing Analysis

Formulating STA as a differentiable process unlocks gradient-based optimization techniques that offer a mathematically sound and interpretable alternative to heuristics. The basic idea is to treat the level-by-level timing propagation as analogous to the forward propagation of neural networks. By running the backward propagation from endpoints back to every timing arc and through delay models, the gradient of objective over design choices can be obtained. Each iteration proceeds in the following steps:

(i) **Loss at endpoints.** Define a loss from endpoint objectives (TNS/WNS).
(ii) **Endpoint → pin.** Map derivatives of endpoint slacks to per-pin AT and $slew$.
(iii) **Pin → arc and physical parameters.** Propagate arrival time gradients along timing arcs to obtain (a) gradients of net arc delays, which backpropagate to parasitics and cell locations; and (b) gradients of cell arc delays and slews, which backpropagate to LUT indices and ultimately to gate sizes.
(iv) **Adjust design.** Accumulate gradients on the design variables (e.g., cell locations in placement, gate sizes in sizing).

Due to the max operators used in timing propagation, directly taking the gradient of a typical STA process leads to a gradient tensor where only pins on the worst negative path have non-zero gradient, which is overly sparse and can lead to ill-conditioned optimization. To solve this problem, *smoothing* techniques like log-sum-exp (LSE) have been applied to differentiable STA to redistribute the backward gradients to more fanin arcs based on relative criticality.

2.3 Problem Formulation

In this paper, we target a challenging physical optimization task as defined in the ICCAD 2025 CAD contest [25]. Given a netlist and a cell library, find an optimal combination of gate sizes s in order to minimize a combination of TNS, WNS, and Power objectives:

$$\begin{aligned} &\min_{s}\ F(s) = \alpha\,|\mathrm{TNS}(s)| \ +\ \beta\,|\mathrm{WNS}(s)| \ +\ \mu\,\mathrm{Power}(s), \\ &\text{subject to}\quad s_g \in \mathcal{L}_g,\ \ \forall g, \end{aligned} \tag{1}$$

where $\mathcal{L}_g$ is the logically equivalent gate sizes of gate g. Here each size is defined as a *combination* of drive strength and threshold voltage. We note the contest setting [25] also allows buffer insertion. However, in this work we focus on gate sizing and insert no extra buffers in combinational circuits. As we show later, by designing an advanced gate sizer, we can remarkably outperform all contest top teams that might have used both gate sizing and buffer insertion.

3 ALGORITHM

Our gate-sizer is based on a differentiable optimization framework similar to [8, 9]. In such a framework, the key questions are (1) how to formulate the problem as a set of numerical variables to optimize,

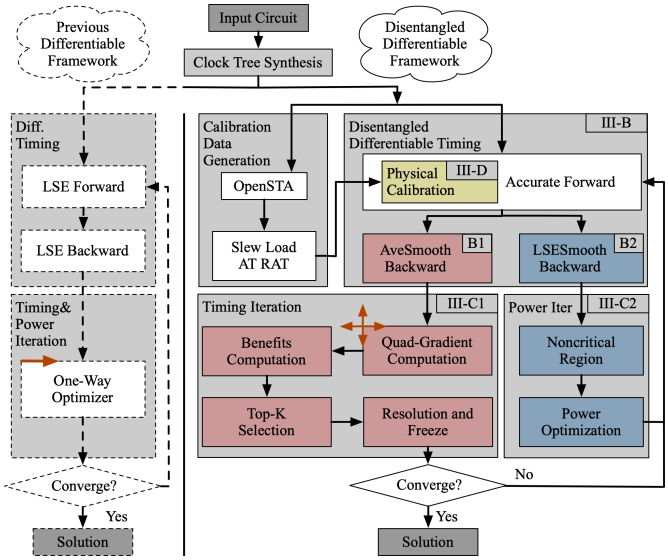

Figure 2: Our proposed framework (right) compared to a vanilla differentiable physical optimization flow (left).

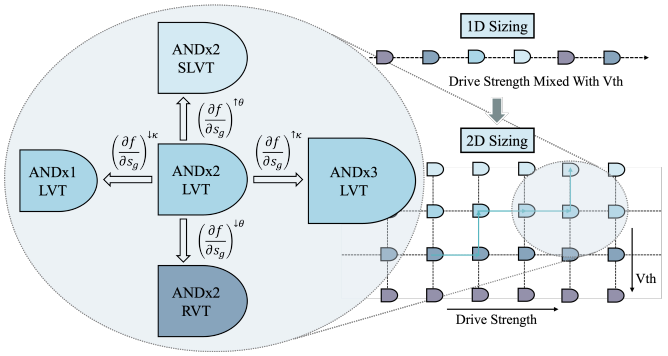

Figure 3: Two-dimensional choice space for gate sizing.

(2) how to obtain gradients for guidance, especially what smoothing strategy to use, (3) how to perform discrete variable updates based on continuous gradients, and (4) how to ensure accurate timing feedback, especially how to achieve robust and effective calibration with external delay annotations. We identify and address challenges on all four aspects in prior work that had hindered the optimization effectiveness of such a multi-objective gate sizing task. The result is a novel differentiable gate sizer that achieves superior results, as shown in Figure 2. The following subsections detail our innovations in four aspects, respectively.

3.1 Two-Dimensional Gate Sizes and Quad Gradients

The raw gate sizing problem optimizes variables $s_g \in \mathcal{L}_g$, which are *choices* instead of numerical values. To formulate choices as numerical values for differentiable optimization, prior works [8, 10] sort all sizes into a 1-dimensional sequence and assign integers 0, 1, 2, ... to the sequence elements in order. However, such sorting and assignment has been shown to introduce non-monotonicity [8] that traps optimization to local optimum. For example, sorting all sizes by fastest-to-slowest typical delay does not always yield largest-to-smallest typical power, and vice versa.

We identify one core issue in sequential assignment: the non-monotonicity lies in the pairwise nature of drive strength and voltage threshold. In fact, all discrete size choices provided by the library

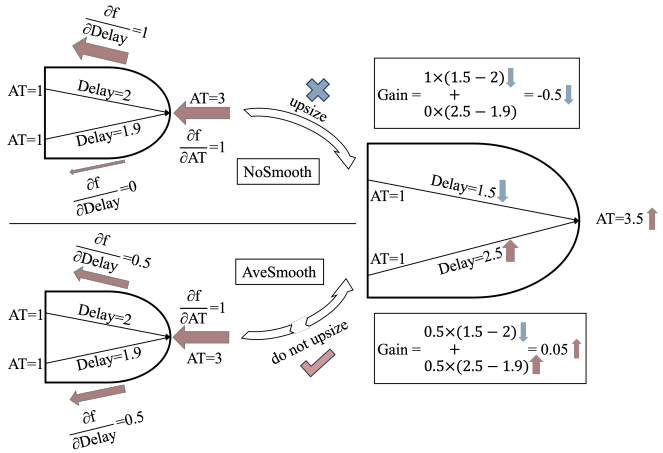

Figure 4: Smoothing is critical for stable timing optimization. Without smoothing (NoSmooth), the full gradient is routed to a single winner fanin arc while a nearly tied competitor receives no gradient, which can worsen the final bottleneck after an update. With smoothing (AveSmooth 3.2.1), the gradient is distributed according to relative criticality, avoiding that failure mode.

spread in a 2-dimensional grid-like space instead of 1 dimensional, as shown in Figure 3. Motivated by this, we assign **2 variables** for every gate size s_g instead of one, representing drive strength κ_g and voltage threshold θ_g respectively. Each optimization iteration generates gradients separately for the two variables. Combined with techniques introduced in the next sections where we calculate gradients differently for upsizing and downsizing, this effectively yields **4 gradients** controlling up- and downgrading strength, as well as up- and downgrading voltage thresholds, respectively.

3.2 Disentangled Upsize and Downsize Gradients

Optimizing WNS and TNS tends to upsize gates to achieve a stronger driver and thus smaller delay, while optimizing power tends to downsize gates because gates with larger sizes (i.e. larger drive strength or smaller voltage threshold) consume a lot of power. A balanced solution assigns larger sizes to critical path cells while downsizing cells on non-critical paths, which is a behavior we try to encapsulate in our differentiable optimizer.

To identify critical paths for timing optimization, differentiable timers employ smoothing techniques like LSE during their backward propagation to spread gradients out. The selection of smoothing strategies and hyperparameters has been directly tied to the delicate balance between critical path focusedness and the prevention of oscillation in WNS/TNS optimization [12], as illustrated in Figure 4.

None of the prior works study the smoothing techniques in *power optimization* yet. We note that power optimization through discrete downsizing should always be applied to cells that are least likely to be on critical paths. As a result, regardless of the timing optimization smoothing strategy, power optimization always requires an aggressive smoothing strategy that spreads gradients as wide as possible.

Motivated by this, we design a framework that back-propagates timing gradients *twice*, separately for timing and power iterations, using different smoothing strategies tailored to their respective needs. Figure 5 visualizes our choices. For timing optimization, we adopt a novel conservative AveSmooth strategy (Section 3.2.1) that concentrates on critical paths. For power optimization, we adopt an

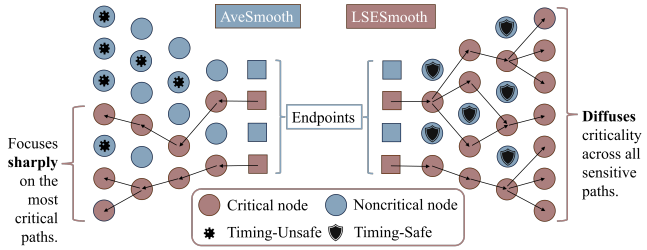

Figure 5: The different smoothing techniques used: AveSmooth (left, for WNS/TNS) and LSESmooth (right, for power).

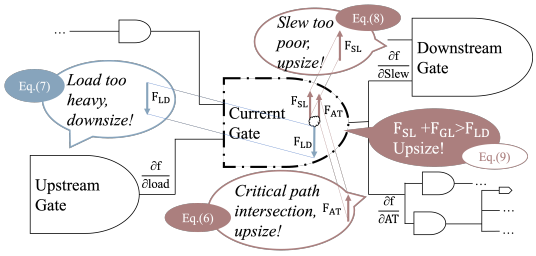

Figure 6: Components of the sizing gradient: AT-driven delay, load via input capacitance, and downstream slew demand.

aggressive LSESmooth strategy with a reasonably-large γ constant (Section 3.2.2) to help us rule out cells that are unsafe to downsize.

3.2.1 AveSmooth for Timing Iteration. For WNS/TNS optimization, we propose a new smoothing strategy AveSmooth which we find to yield better solution quality than LSE-based smoothing. The basic idea is to spread gradients evenly to fanin arcs that are close to the maximum one up to a fixed percentage $\epsilon = 0.01$. This is a simple yet effective way to spread gradients while maintaining our focus on critical paths.

$$\mathcal{N}_\varepsilon(v) = \Big\{ u \;\Big|\; AT(u) + d_{u\to v} \geq (1-\varepsilon)\max_{u'}\big(AT(u') + d_{u'\to v}\big)\Big\}, \tag{2}$$

$$w^{\text{ave}}_{u\to v}(\text{AT}) = \begin{cases} 1/|\mathcal{N}_\varepsilon(v)|, & u \in \mathcal{N}_\varepsilon(v), \\ 0, & \text{otherwise}, \end{cases} \tag{3}$$

$$\frac{\partial f}{\partial AT(u)} \;+\!=\; w^{\text{ave}}_{u\to v}(\text{AT}) \cdot \frac{\partial f}{\partial AT(v)}. \tag{4}$$

3.2.2 LSESmooth for Power Iteration. For power optimization, we adopt LSESmooth [12, 28] with a reasonably-large $\gamma = 1$ constant. This rules out near-critical cells aggressively to ensure no timing degradation when downsizing cells for power benefit.

$$w^{\text{lse}}_{u\to v} = \frac{\exp\big((AT(u)+d_{u\to v})/\gamma\big)}{\sum_{u'}\exp\big((AT(u')+d_{u'\to v})/\gamma\big)}, \qquad \frac{\partial f}{\partial AT(u)} \;+\!=\; w^{\text{lse}}_{u\to v}\cdot\frac{\partial f}{\partial AT(v)}. \tag{5}$$

3.3 Upsizing/Downsizing with Gradient Feedback

The gradients generated by backpropagation carry rich and interpretable information which we use to instruct upsizing and downsizing of cells. This section details the interpretable gradient in the context of gate sizing, and our strategies on updating discrete variables.

We use back-propagation to obtain gradients of κ_g and θ_g using chain rule of partial derivatives. This naturally splits the gradients on size variables to 3 components during equation derivations, as depicted in Figure 6:

(i) *Demand in reducing the cell delay itself.* The gradients on arrival time are back-propagated and assigned to individual cell arcs. A large delay gradient indicates a critical cell arc, i.e., it is on one or multiple critical paths or affecting many individual endpoints inside TNS.

$$\left(\frac{\partial f}{\partial s_g}\right)_{\text{AT}} = \sum_{u\in\text{fanin}(v)} \underbrace{\frac{\partial f}{\partial AT(v)}}_{\text{global}} \cdot w^{\text{ave}}_{u\to v}(\text{AT}) \cdot \frac{\partial\, LUT^{u\to v}_{delay}}{\partial s_g}. \tag{6}$$

(ii) *Pressure from upstream driver to reduce capacitive load.* Input pin capacitance of the current cell contributes to the load driven by the upstream cell. Thus, the demand to lower upstream cell delay encourages the downsize of the current cell.

$$\left(\frac{\partial f}{\partial s_g}\right)_{\text{load}} = \sum_{u\in\text{fanin}(v)} \underbrace{\frac{\partial f}{\partial Cap_u}}_{\text{upstream}} \cdot \frac{\partial Cap_u}{\partial s_g}. \tag{7}$$

(iii) *Pressure from downstream receiver to reduce its output slew.* Similarly, the delay of a downstream cell benefits from the potential upsize of current cell since large cells output smaller slews.

$$\left(\frac{\partial f}{\partial s_g}\right)_{slew} = \sum_{u\in\text{fanin}(v)} \underbrace{\frac{\partial f}{\partial slew(v)}}_{\text{downstream}} \cdot w^{\text{ave}}_{u\to v}(slew) \cdot \frac{\partial\, LUT^{u\to v}_{slew}}{\partial s_g}. \tag{8}$$

Aggregating the three sources yields

$$\frac{\partial f}{\partial s_g} = \left(\frac{\partial f}{\partial s_g}\right)_{\text{AT}} + \left(\frac{\partial f}{\partial s_g}\right)_{\text{load}} + \left(\frac{\partial f}{\partial s_g}\right)_{slew}. \tag{9}$$

The partial derivatives $\partial LUT/\partial s_g$ and $\partial Cap/\partial s_g$ are not taken with respect to a continuous parameter, but along discrete library moves. Let gate size $s_g = (\kappa_g, \theta_g)$ where κ_g denotes ordered integer drive strength and θ_g denotes ordered integer voltage threshold. For convenience, we use $\uparrow \kappa_g$, $\downarrow \kappa_g$, $\uparrow \theta_g$, and $\downarrow \theta_g$ to denote the 4 possible update actions. For any quantity $X \in \{LUT_{delay}, LUT_{slew}, Cap\}$ and any candidate action a that maps the current state s_g to a neighboring state s_g^a, we approximate the directional derivative along a by a one-sided finite difference $(\partial X/\partial s_g)^a \approx X(s_g^a) - X(s_g)$.

3.3.1 Timing Update. Using the chain-rule structure in (6)–(8) with these finite differences, we form the per-action directional estimate; $(\partial f/\partial s_g)^a$ denotes the directional derivative under action a.

$$\left(\frac{\partial f}{\partial s_g}\right)^{\uparrow\kappa_g}, \quad \left(\frac{\partial f}{\partial s_g}\right)^{\downarrow\kappa_g}, \quad \left(\frac{\partial f}{\partial s_g}\right)^{\uparrow\theta_g}, \quad \left(\frac{\partial f}{\partial s_g}\right)^{\downarrow\theta_g}.$$

These four directional derivatives, corresponding to the four primitive actions in the 2D drive-strength (κ) and threshold-voltage (θ) space, are collectively referred to as the quad-gradient. We convert them into benefits using the sign convention:

$$\psi_g^{\uparrow\kappa} = -\left(\frac{\partial f}{\partial s_g}\right)^{\uparrow\kappa_g}, \quad \psi_g^{\uparrow\theta} = -\left(\frac{\partial f}{\partial s_g}\right)^{\uparrow\theta_g}, \tag{10}$$

$$\psi_g^{\downarrow\kappa} = +\left(\frac{\partial f}{\partial s_g}\right)^{\downarrow\kappa_g}, \quad \psi_g^{\downarrow\theta} = +\left(\frac{\partial f}{\partial s_g}\right)^{\downarrow\theta_g}. \tag{11}$$

In practice, upsizing ($\uparrow \kappa$) can degrade timing by loading upstream drivers, whereas downsizing ($\downarrow \kappa$) may help; this motivates evaluating all four directions and explains the baseline's negative "optimization" entries, highlighting the difficulty of gate sizing.

Algorithm 1: GPU-parallel Top-K quad-gradient update

Input: Quad-gradient scores ψ_g^d for $d \in \{\uparrow \kappa, \downarrow \kappa, \uparrow \theta, \downarrow \theta\}$; validity/freeze masks; budget K
Output: Updated discrete gate states s_g

Build candidate list $\mathcal{C} = \{(g, d, \psi_g^d) \mid d \text{ valid}, \ \psi_g^d > 0, \ g \text{ not frozen}\}$ in parallel
$\Psi \leftarrow$ vector of all ψ_g^d in $\mathcal{C}$
idx $\leftarrow$ $\texttt{topk}(\Psi, K)$ on GPU
$\mathcal{S} \leftarrow \emptyset$; mark all gates as unused
foreach $i \in$ idx *(in descending order)* **do**
  $(g, d, \psi) \leftarrow \mathcal{C}[i]$
  if g *already used* **then**
    continue
  add (g, d) to $\mathcal{S}$ and mark g as used
Update states for $(g, d) \in \mathcal{S}$ in parallel; freeze g at s_g^{best} if oscillating $(s_g \rightleftarrows s'_g)$

Large-magnitude gradients usually correspond to gates that affect many timing paths, so prioritizing them yields more global improvement while changing fewer gates; hence we use a Top-K strategy that updates only gates with the largest gradients.

Collect all candidate tuples into $\mathcal{C} = \{(g, d, \psi_g^d) \mid d \in \{\uparrow \kappa, \downarrow \kappa, \uparrow \theta, \downarrow \theta\}\}$ and perform a single GPU-parallel Top-K update with budget K; the procedure is summarized in Alg. 1. At termination, we roll back to the best state $S^\star$ if the final objective exceeds $f^\star$.

3.3.2 Power Update. Under LSESmooth, we treat locations satisfying $|\partial f / \partial AT| \le \tau$ as timing-safe. We then apply power-reducing actions there, either by decreasing drive strength ($\downarrow \kappa$) or by raising Vth, for example from LVT to RVT. Every few timing iterations, we run one LSE-power step. It first uses the LSESmooth backward pass to find timing-safe locations and then applies a subset of these actions to save power without hurting timing.

Traditional RAT-based methods see only local slack and are blind to a gate's global impact scope. A cell in a large fanout cone may look locally timing-safe, yet downsizing it compresses the optimization space of many other gates. In contrast, LSESmooth produces inherently global gradients: cells that influence many gates and endpoints naturally accumulate larger magnitudes, discouraging premature downsizing of these high-impact locations.

3.4 Multi-Level Physical Calibration

Prior work on differentiable timing calibration [23] mostly stops at correcting arc delays, which cannot provide sufficiently accurate physical information for physical-level optimization. For gate sizing in particular, computing the delay and propagated slew of a gate requires both the input slew and the output load, and only well-calibrated slew and load can ensure that the benefit of a sizing change is evaluated faithfully. Moreover, the backpropagation of AT gradients relies on accurate AT values, and only precise AT can steer gradients toward the locations that truly require optimization. To this end, we develop a Multi-Level Physical Calibration scheme (Figure 7) that, beyond conventional delay calibration, also calibrates slew and load, enabling more precise sizing while greatly reducing harmful updates.

We first run our timer once with an uncalibrated model to obtain slew, load, and AT, and divide the reference STA results by these values to derive separate calibration ratios for each quantity. In subsequent iterations, we multiply our own predictions by the corresponding ratios. The ratios are computed layer by layer: if one layer deviates, it is corrected before propagating to the next layer, so each layer is evaluated on already calibrated inputs. This hierarchical scheme keeps the per-layer ratios close to one and allows the calibration effect to persist over many optimization iterations.

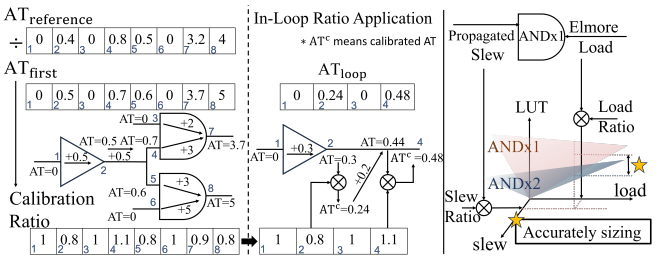

Figure 7: Multi-Level Physical Calibration for Differentiable Sizing. Left: calibration ratios are derived by comparing reference arrival times from OpenSTA against the first differentiable estimate. Middle: these ratios correct later loop values by multiplication. Right: calibrated slew and load place each gate at an accurate operating point on its LUT surface, enabling reliable evaluation of sizing benefits.

4 EXPERIMENTAL RESULTS

4.1 Experimental Setup

We conducted all experiments on a Linux server with an Intel Xeon Silver 4214 CPU and 256GB memory. The optimization routines were accelerated using one NVIDIA RTX A6000 GPU with 48 GB VRAM. To ensure a fair comparison with other methods, we followed the setting of ICCAD 2025 Contest [25] and limited all experiments to use 8 CPU threads.

In our experiments, we set $\epsilon = 0.01$, $\gamma = 1$, and $\tau = 10^{-6}$. The number of gate updates per iteration (K) is dynamically adjusted during the optimization (ranging from 10 to 500) to balance convergence speed with optimization stability.

4.2 Comparison with ICCAD 2025 Contest Flows

We evaluate our framework on all benchmarks of the ICCAD 2025 Contest [25]. We obtain the contest official evaluation reports from the 1st and 2nd winner teams for comparison. We cannot compare with the 3rd winner team due to the unavailability of their report. The contest rules allow a combination of three optimization techniques: gate sizing, buffering, and cell relocation. The evaluation flow of our approach follows exactly the same procedure as that in the contest. We apply our core optimization engine, which relies exclusively on gate sizing. After our optimization, we apply a mandatory legalization step to produce a valid and legal solution as required by the contest. This step is run for compliance and incurs a non-trivial performance penalty. This fully compliant flow allows a direct comparison of our sizing-centric engine against the multi-technique flows of the baselines. The runtime for the 1st and 2nd winner teams was evaluated under a different Linux machine with similar CPUs, but stronger GPU, i.e., 8 Xeon CPUs and one A100 GPU. As the hardware differs, the runtime comparison is just for reference.

As shown in Table I, our approach can significantly reduce |TNS| by 51.8% on average, which is 40.4%*pt* and 43.1%*pt* better than the contest winner teams. Moreover, the contest winner teams achieve timing optimization at the cost of noticeable power increase (5.2% and 14.6%), while, by contrast, our approach can even reduce power by 11.0%. We ascribe such remarkable benefits to the disentangled optimization of timing and power mentioned in Sec. 3.3.2, which not only optimizes timing by careful sizing up operations in timing-critical regions, but

Table 1: Comparison with the top two teams of the ICCAD 2025 Gate Sizing Contest.

Design[1]	#Cells	Init. TNS(ps)	Δ|TNS| (%)[2](↓)			Init. Power(W)	ΔPower (%) (↓)			Runtime (s) (↓)		
			1st	2nd	ours		1st	2nd	ours	1st	2nd	ours
NV_c	167.8K	-3.52E+07	-31.9	-30.1	**-41.5**	1.63E+00	+8.6	-1.2	**-2.5**	518	209	**180**
ac97	7.8K	-1.98E+05	+16.5	-7.9	**-49.2**	1.36E-01	**-56.5**	-13.2	-44.8	**5**	13	30
aes	4.4K	-6.83E+03	-58.4	+30.3	**-63.4**	3.74E-02	+24.1	+9.9	**+1.3**	**8**	11	31
cipher	11.6K	-1.40E+04	-38.3	-37.8	**-78.2**	7.84E-02	-0.8	**-2.7**	+0.4	**10**	11	42
des3	2.2K	-2.42E+04	0.0	-2.5	**-27.7**	1.25E-02	0.0	**-11.2**	+15.2	450	**8**	33
mc_top	5.4K	-6.68E+04	+0.0	-8.6	**-37.2**	1.29E-02	-2.3	**-4.7**	+10.9	865	**13**	53
pool	182.9K	-1.14E+07	-1.2	+0.7	**-3.1**	3.96E+00	+248.5	+228.3	**+0.3**	113270[3]	**283**	299
netcard	298.4K	-3.46E+08	+147.0	+55.4	**-100.0**	1.68E+00	**-95.1**	+1.2	-90.7	182604[3]	741	**169**
pci	12.1K	-3.89E+05	-2.0	-13.2	**-90.4**	1.28E-01	-49.2	-3.9	**-60.2**	**6**	13	41
tv80s	3.7K	-7.67E+04	+0.0	-8.0	**-20.9**	1.11E-02	**-0.9**	+3.6	+21.6	641	**9**	41
ariane	105.7K	-3.13E+03	**-99.8**	-91.5	-97.8	4.55E-02	-14.3	**-22.0**	0.0	942	**47**	339
des	2.3K	-2.05E+03	**-9.5**	-5.8	-7.1	1.58E-02	0.0	**-8.9**	+5.7	**4**	7	36
fpu	22.2K	-5.87E+03	**-70.8**	+5.4	-57.0	2.84E-01	+49195.8[3]	+46378.9[3]	**0.0**	5790	**34**	116
Ave.			-11.4	-8.7	**-51.8**		+5.2	+14.6	**-11.0**	840	108	108

[1] Abbr: NV_c (NV_NVDLA_partition_c), ac97_top (ac97), cipher (aes_cipher_top), pool (mempool_tile_wrap), netcard (netcard_fast), pci (pci_bridge32).
[2] WNS is omitted since the contest metric only uses TNS.
[3] Extreme reported values that are kept for completeness but treated as outliers and excluded from the Ave. row averages.

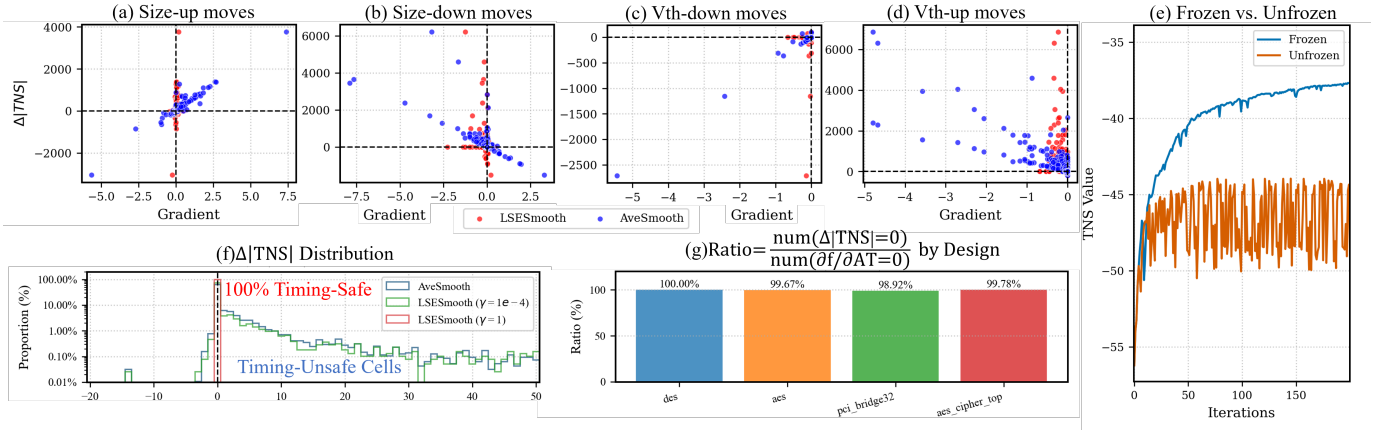

Figure 8: Empirical analysis of our gradients and update policies. (a–e) Results on `pci_bridge32`: correlation between quad-gradient scores and Δ|TNS|, the effect of AveSmooth versus LSESmooth on gradient distributions, and TNS trajectories with and without the freeze strategy in Algorithm 1. (f–g) Results on `des`: Δ|TNS| of power-reduction moves inside timing-safe regions defined by $|\partial f/\partial \mathrm{AT}| \le \tau$ under different smoothing schemes.

also reduces power by sizing-down operations in non-critical regions. For reference, our runtime is also very competitive even on a weaker GPU compared with the contest winner teams.

4.3 Empirical Analysis of Gradients, Freezing, and Timing-Safe Regions

We empirically examine how our gradients and update policies behave on representative ICCAD 2025 benchmarks. On `pci_bridge32` with an ideal clock setting (Figs. 8a–e), the quad-gradient scores are strongly correlated with the true change in |TNS| for individual sizing moves. AveSmooth also produces a much sharper and cleaner gradient distribution than LSESmooth. The TNS trajectories show that the freeze strategy in Algorithm 1 suppresses oscillation and steadily lowers |TNS|, whereas the no-freeze variant oscillates at a worse level. On `des` (Figs. 8f–g), we validate our timing-safe region for power optimization, defined by locations with $|\partial f/\partial \mathrm{AT}| \le \tau$. With LSESmooth at $\gamma = 1$, all down-sizing moves in this region keep |TNS| unchanged. By contrast, AveSmooth and LSESmooth with $\gamma = 10^{-4}$ produce many moves that worsen |TNS|. This confirms that the large-γ LSESmooth branch reliably screens noncritical regions for power recovery.

5 CONCLUSION

In this paper, we propose a disentangled differentiable gate sizing framework for timing-power co-optimization. By defining quad-gradients for discrete drive × Vth space, we enable effective optimization in a 2D discrete space with differentiable approaches. We further decouple the optimization of timing and power with two backward branches to identify timing critical and non-critical regions. Experimental results on the ICCAD 2025 contest benchmarks demonstrate that our framework can achieve 40.4%*pt* better TNS improvement and 16.2%*pt* better total power change compared with the contest winner teams.

REFERENCES

[1] J. Bhasker and R. Chadha. *Static Timing Analysis for Nanometer Designs: A Practical Approach*. Springer Publishing Company, Incorporated, 1st edition, 2009. ISBN 0387938192.

[2] C.-P. Chen, C. C. Chu, and D. Wong. Fast and exact simultaneous gate and wire sizing by lagrangian relaxation. In *Proceedings of the 1998 IEEE/ACM international conference on Computer-aided design*, pages 617–624, 1998.

[3] C.-K. Cheng, C. Holtz, A. B. Kahng, B. Lin, and U. Mallappa. Dagsizer: A directed graph convolutional network approach to discrete gate sizing of vlsi graphs. *ACM Transactions on Design Automation of Electronic Systems*, 28(4):1–31, 2023.

[4] D. Chinnery and A. Sharma. Integrating lr gate sizing in an industrial place-and-route flow. In *Proceedings of the 2022 International Symposium on Physical Design*, pages 39–48, 2022.

[5] 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*, pages 801–806, 2005.

[6] O. Coudert. Gate sizing for constrained delay/power/area optimization. *IEEE Transactions on Very Large Scale Integration (VLSI) Systems*, 5(4):465–472, 2002.

[7] S. Daboul, N. Hähnle, S. Held, and U. Schorr. Provably fast and near-optimum gate sizing. *IEEE transactions on computer-aided design of integrated circuits and systems*, 37(12):3163–3176, 2018.

[8] Y. Du, Z. Guo, Y. Lin, R. Wang, and R. Huang. Fusion of global placement and gate sizing with differentiable optimization. In *ICCAD*. ACM, 2024.

[9] Y. Du, Z. Guo, Y. Hsu, Z. Xiong, S. Kim, D. Pan, R. Wang, and Y. Lin. Addressing continuity and expressivity limitations in differentiable physical optimization: A case study in gate sizing. In *2025 International Symposium of EDA (ISEDA)*, 2025.

[10] Y. Du, Z. Guo, R. Wang, and Y. Lin. Differentiable physical optimization. In *ICCAD*, 2025.

[11] J. P. Fishburn and A. E. Dunlop. Tilos: A posynomial programming approach to transistor sizing. In *The Best of ICCAD: 20 Years of Excellence in Computer-Aided Design*, pages 295–302. Springer, 2003.

[12] Z. Guo and Y. Lin. Differentiable-timing-driven global placement. In *DAC*, page 1315–1320, 2022. doi: 10.1145/3489517.3530486.

[13] S. Held. Gate sizing for large cell-based designs. In *2009 Design, Automation & Test in Europe Conference & Exhibition*, pages 827–832. IEEE, 2009.

[14] J. Hu, A. B. Kahng, S. Kang, M.-C. Kim, and I. L. Markov. Sensitivity-guided metaheuristics for accurate discrete gate sizing. In *Proceedings of the International Conference on Computer-Aided Design*, pages 233–239, 2012.

[15] S. Hu, M. Ketkar, and J. Hu. Gate sizing for cell library-based designs. In *Proceedings of the 44th annual Design Automation Conference*, pages 847–852, 2007.

[16] S. Joshi and S. Boyd. An efficient method for large-scale gate sizing. *IEEE Transactions on Circuits and Systems I: Regular Papers*, 55(9):2760–2773, 2008.

[17] A. B. Kahng, S. Kang, H. Lee, I. L. Markov, and P. Thapar. High-performance gate sizing with a signoff timer. In *2013 IEEE/ACM International Conference on Computer-Aided Design (ICCAD)*, pages 450–457. IEEE, 2013.

[18] W. Li, Y. Kukimoto, G. Servel, I. Bustany, and M. E. Dehkordi. Calibration-based differentiable timing optimization in non-linear global placement. In *ISPD*, 2024.

[19] 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 TCAD*, 40(4):748–761, 2021. doi: 10.1109/TCAD.2020.3003843.

[20] Y. Liu and J. Hu. Gpu-based parallelization for fast circuit optimization. *ACM Transactions on Design Automation of Electronic Systems (TODAES)*, 16(3):1–14, 2011.

[21] J. Lu, P. Chen, C.-C. Chang, L. Sha, D. J.-H. Huang, C.-C. Teng, and C.-K. Cheng. eplace: Electrostatics-based placement using fast fourier transform and nesterov's method. *ACM Trans. Des. Autom. Electron. Syst.*, 20(2), mar 2015. ISSN 1084-4309.

[22] 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)*, pages 733–738. IEEE, 2021.

[23] 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 62nd ACM/IEEE Design Automation Conference (DAC)*, pages 1–7, 2025. doi: 10.1109/DAC63849.2025.11132858.

[24] Y.-C. Lu, K. Kunal, G. Pradipta, R. Liang, R. Gandikota, and H. Ren. Lego-size: Llm-enhanced gpu-optimized signoff-accurate differentiable vlsi gate sizing in advanced nodes. In *ISPD*, 2025.

[25] 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 *ICCAD*, 2025.

[26] D. Mangiras, D. Chinnery, and G. Dimitrakopoulos. Task-based parallel programming for gate sizing. *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, 42(4):1309–1322, 2022.

[27] S. Nath, G. Pradipta, C. Hu, T. Yang, B. Khailany, and H. Ren. Transsizer: A novel transformer-based fast gate sizer. In *Proceedings of the 41st IEEE/ACM International Conference on Computer-Aided Design*, pages 1–9, 2022.

[28] W. C. Naylor, R. Donelly, and L. Sha. Non-linear optimization system and method for wire length and delay optimization for an automatic electric circuit placer, Oct. 9 2001. US Patent 6,301,693.

[29] W. Ning. Strongly np-hard discrete gate-sizing problems. *IEEE transactions on computer-aided design of integrated circuits and systems*, 13(8):1045–1051, 1994.

[30] M. M. Ozdal, C. Amin, A. Ayupov, S. Burns, G. Wilke, and C. Zhuo. The ispd-2012 discrete cell sizing contest and benchmark suite. In *Proceedings of the 2012 ACM international symposium on International Symposium on Physical Design*, pages 161–164, 2012.

[31] M. M. Ozdal, S. Burns, and J. Hu. Algorithms for gate sizing and device parameter selection for high-performance designs. *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, 31(10):1558–1571, 2012.

[32] P. Pham and J. Chung. Agd: A learning-based optimization framework for eda and its application to gate sizing. In *2023 60th ACM/IEEE Design Automation Conference (DAC)*, pages 1–6. IEEE, 2023.

[33] A. Sharma, D. Chinnery, S. Bhardwaj, and C. Chu. Fast lagrangian relaxation based gate sizing using multi-threading. In *2015 IEEE/ACM International Conference on Computer-Aided Design (ICCAD)*, pages 426–433. IEEE, 2015.

[34] A. Sharma, D. Chinnery, T. Reimann, S. Bhardwaj, and C. Chu. Fast lagrangian relaxation-based multithreaded gate sizing using simple timing calibrations. *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, 39(7): 1456–1469, 2019.

[35] B.-Y. Wu, R. Liang, G. Pradipta, A. Agnesina, H. Ren, and V. A. Chhabria. 2024 iccad cad contest problem c: Scalable logic gate sizing using ml techniques and gpu acceleration. In *Proceedings of the 43rd IEEE/ACM International Conference on Computer-Aided Design*, pages 1–5, 2024.

[36] J. Xie and C. R. Chen. Lookup table based discrete gate sizing for delay minimization with modified elmore delay model. In *Proceedings of the 25th edition on Great Lakes Symposium on VLSI*, pages 361–366, 2015.

[37] Y. Ye, P. Xu, L. Ren, T. Chen, H. Yan, B. Yu, and L. Shi. Learning-driven physically-aware large-scale circuit gate sizing. *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, 2024.