FROM CRITICAL PATHS TO CRITICAL CELLS: LEARNING FINE-GRAINED TIMING ATTRIBUTION FOR CHIP DESIGN

Anonymous authors
Paper under double-blind review

ABSTRACT

Identifying which components of a system to improve is difficult when supervision is available only for aggregated outcomes. Timing optimization in chip design presents this challenge: static timing analysis (STA) identifies timing-critical paths, while placement refinement is made on cells whose contributions to a path's timing degradation may differ. Identifying a critical path does not by itself reveal the cells whose optimization would most improve timing. We study **timing-critical cell identification**, the task of learning cell-level timing guidance from path-level observations without cell-level annotations. We introduce CritiCell, a multiple instance learning framework that represents timing paths as bags of cells and learns cell scores from their aggregate timing behavior. We also utilize these scores to guide both constraint-driven and objective-driven placement in the `ICCAD2015` benchmark suite. Compared with coarse path-level guidance, our fine-grained CritiCell improves worst negative slack (WNS) and total negative slack (TNS) by 2.87% and 10.74% in constraint-driven placement, and by 4.08% and 10.56% in objective-driven placement. These results show that coarse supervision can produce fine-grained guidance useful for downstream optimization tasks.

1 INTRODUCTION

Electronic Design Automation (EDA) tools play a fundamental role in modern chip design by enabling designers to cope with the rapidly increasing complexity and scale of integrated circuits (MacMillen et al., 2000; Markov et al., 2012). Across the EDA workflow, a central objective is to optimize physical and performance metrics, among which **timing performance** is particularly critical. Timing performance determines the maximum operating frequency of a chip and directly reflects whether signals can propagate reliably within prescribed timing constraints (Rabaey et al., 2002; Wang et al., 2009).

To evaluate timing quality, **Static Timing Analysis (STA)** has become a standard and indispensable tool in digital chip design (Bhasker & Chadha, 2009). Given a circuit implementation and a target clock period, STA analyzes signal propagation delays without relying on dynamic simulation and reports potential timing violations. A key output of STA is a collection of timing paths, each associated with a slack value that quantifies how well the path meets timing constraints. Paths with negative slack are identified as timing-critical and are the primary focus of timing optimization efforts.

Because STA provides effective timing evaluation, it serves as a major source of guidance for timing-aware chip design (OpenSTA, 2023; Fu et al., 2024; Shi et al., 2025). In practice, STA-derived timing information—such as timing-critical paths and their slack values—is widely used to guide timing optimization throughout the design process (Guo & Lin, 2022; Liao et al., 2022; Lin et al., 2024; Fu et al., 2024; Shi et al., 2025).

However, using this feedback to guide timing optimization exposes a granularity mismatch between path-level timing assessment and cell-level optimization decisions. Cells on the same critical path need not contribute equally to that path's timing degradation, yet identifying a violating path does not directly reveal its major contributing cells (Bhasker & Chadha, 2009; OpenSTA, 2023). We refer to these major contributors as **timing-critical cells**. Existing approaches incorporate path-level timing criticality into optimization objectives or constraints (Liao et al., 2022; Bustany et al., 2023; Shi et al.,

Reviewers: please read the Reviewer Guidelines (iclr.cc/Conferences/2027/ReviewerGuidelines) and the AI Policy for Reviewers (iclr.cc/Conferences/2027/AIPolicyForReviewers). If you used AI to expand, edit, or polish your review, please provide the input text to the LLM. Better still, consider skipping the LLM and submitting your original text: we, and the authors, are much more interested in your unedited thoughts than in what an LLM has to say. AI-assisted or not, you are putting your name and reputation behind your review: LLM-generated falsehoods, hallucinations or misrepresentations are subject to disciplinary action, which may include desk-rejecting all papers you have authored.

2025), but do not explicitly distinguish the contributions of individual cells to timing degradation on the same path. Addressing this granularity mismatch can enable more targeted timing optimization, focusing effort on local bottlenecks while limiting unnecessary perturbations to other parts of the design.

To address this granularity mismatch, we propose **CritiCell**, a weakly supervised learning framework that bridges path-level feedback and cell-level optimization guidance. CritiCell represents paths as bags of cells under a **multiple instance learning (MIL)** formulation (Wang et al., 2018; Waqas et al., 2024) and learns cell-level criticality scores from their collective timing behavior. By using STA-derived path labels as supervision, the framework learns to identify timing-critical cells without cell-level annotations, providing fine-grained guidance for timing optimization.

We evaluate the fine-grained guidance learned by CritiCell through two representative timing-aware placement applications. Specifically, we show how the identified timing-critical cells can (i) guide constraint construction in clustering-based placement (Fogaca et al., 2020; Lu et al., 2021; 2022; Kahng et al., 2024) and (ii) be incorporated into the objective function of Efficient-TDP (Shi et al., 2025)—an existing timing-driven placer. On the ICCAD15 benchmarks, timing-critical-cell-guided clustering-based placement achieves average gains of **2.87%** in WNS and **10.74%** in TNS over the path-level timing-aware TritonPart baseline, while enhancing Efficient-TDP with timing-critical cells yields improvements of **4.08%** in WNS and **10.56%** in TNS over Efficient-TDP. These results demonstrate that cell-level timing information learned from path-level supervision can effectively guide chip optimization.

While we instantiate and evaluate our approach in timing-aware placement, it addresses a broader learning problem: deriving fine-grained decision guidance from coarse-grained supervision when component-level annotations are unavailable. In our setting, path-level timing observations supervise the learning of cell-level scores. This formulation may be relevant to other applications in which outcomes are measured for groups of components, but decisions must be made about individual components.

2 PRELIMINARIES

2.1 MULTIPLE INSTANCE LEARNING

Multiple instance learning (MIL) is a weakly supervised paradigm where training data are given as bag–label pairs $\{(B_i, y_i)\}_{i=1}^{N}$ (Wang et al., 2018; Waqas et al., 2024). Each bag $B_i = \{x_{i1}, \ldots, x_{iM_i}\}$ contains multiple instances. In the standard binary classification setting of MIL, a bag is positive if it contains at least one positive instance, and negative otherwise; only bag-level labels are observed during training.

From the perspective of classification granularity, MIL algorithms can be broadly categorized into bag-level algorithm, which aims to predict bag-level labels, and instance-level algorithm, which aims to infer instance-level labels (Waqas et al., 2024).

mi-Net is a representative instance-level MIL algorithm (Wang et al., 2018; Waqas et al., 2024). Given a bag of instances, mi-Net applies a shared instance-level predictor to each instance independently, producing a prediction score for every instance. The instance-level predictions are then aggregated using a permutation-invariant pooling operator to obtain a bag-level prediction. This bag-level prediction is compared with the observed bag-level label during training, enabling instance-level inference without requiring instance-level annotations.

2.2 BASICS OF CHIPS

A digital chip mainly consists of a collection of cells connected by nets. **A cell** represents a basic building block of digital logic, such as a logic gate or a flip-flop. **A net** represents an electrical connection that carries signals between cells. Each cell exposes a set of **pins** that serve as interfaces for signal input and output, referred to as **input pins** and **output pins**, respectively. A net typically connects the output pin of one cell to the input pins of one or more other cells, enabling signal propagation across the circuit. **A netlist** of a chip specifies the circuit interconnections by listing, for

each net, the pins it connects. It serves as the input for downstream physical design steps such as placement and routing.

Placement assigns each cell a location on the chip canvas. It is commonly divided into **global placement** and **detailed placement**. We mainly discuss global placement, which computes approximate cell locations in a continuous layout space and is typically formulated as an optimization problem.[1]

2.3 Static Timing Analysis

Static Timing Analysis (STA) is a standard technique to evaluate timing quality without dynamic simulation (Bhasker & Chadha, 2009). Given a circuit and a target clock period, STA analyzes signal propagation delays and identifies timing violations.

A **timing path** p is an ordered sequence of pins through which a signal propagates within one clock cycle, typically from a launch point (a flip-flop output or a primary input) to a capture point (a flip-flop input or a primary output). For each path, STA computes the **arrival time** $\mathrm{AT}(p)$ (i.e., the signal propagation time along the path) and the **required arrival time** $\mathrm{RAT}(p)$ (i.e., the maximum allowable signal propagation time under the clock constraint), and defines the **slack** as

$$\mathrm{Slack}(p) = \mathrm{RAT}(p) - \mathrm{AT}(p).$$

Negative slack indicates a timing violation, and paths with negative slack are referred to as **timing-critical paths**.

Two standard timing metrics are **worst negative slack (WNS)**, i.e., the minimum slack over all endpoints/paths, and **total negative slack (TNS)**, i.e., the sum of negative slacks across all violating endpoints/paths. Values closer to zero indicate better timing performance.

2.4 Timing-Aware Chip Design

Timing-aware chip design methods leverage STA feedback to improve timing quality and, from an optimization perspective, commonly adopt **constraint-driven** or **objective-driven** integration (Fu et al., 2024; Shi et al., 2025; Guo & Lin, 2022; Liao et al., 2022; Lu et al., 2022; Kahng et al., 2024; Swartz & Sechen, 1995; Chang et al., 2002; Bustany et al., 2023).

Constraint-Driven Timing Optimization Constraint-driven timing optimization leverages STA feedback by converting timing information into explicit structural constraints or guidance signals that steer downstream optimizers. A representative example is *clustering-based placement* (Lu et al., 2022; Kahng et al., 2024), which aims to improve timing quality (often jointly with PPA metrics such as power and area).

Objective-Driven Timing Optimization Objective-driven timing optimization incorporates STA feedback by augmenting the placement objective with timing-related terms. A representative example is Efficient-TDP (Shi et al., 2025), which adds a *pin-to-pin attraction* loss $\mathrm{PP}(\cdot)$ on STA-identified timing-critical paths to encourage their interconnects to be shortened during placement.

Efficient-TDP periodically runs STA and maintains a set P of *pin pairs* (i, j) extracted from critical paths, where each pair corresponds to two path-adjacent pins from consecutive cells. Each pin pair is assigned a weight that is updated based on the path slack (normalized by WNS):

$$w(i,j) = \begin{cases} w_0, & (i,j) \notin P, \\ w(i,j) + w_1 \cdot (\mathrm{slack}/\mathrm{WNS}), & \text{otherwise.} \end{cases} \tag{1}$$

The attraction loss is then defined as

$$\mathrm{PP}(\mathbf{x}, \mathbf{y}) = \sum_{(i,j)\in P} w(i,j) \cdot Q(i,j), \tag{2}$$

where $Q(i, j)$ measures squared Euclidean distance between pins i and j.

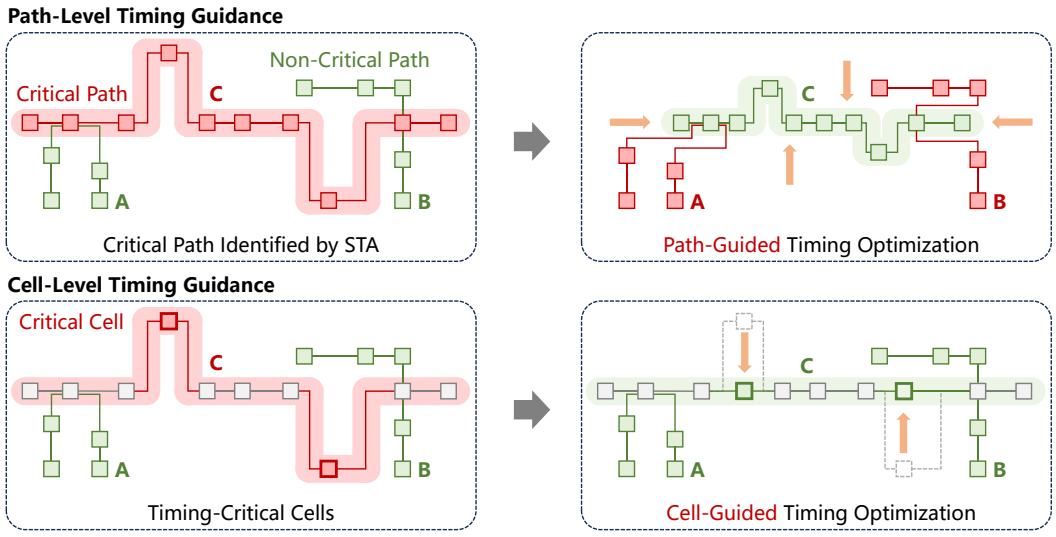


Figure 1: Potential benefits of cell-level timing guidance for addressing the granularity mismatch, illustrated using placement as an example. Path-level guidance (top) distributes optimization effort across critical path C, potentially perturbing other paths such as A and B. Cell-level guidance (bottom) prioritizes timing-critical cells within C, which can help target local bottlenecks while limiting unnecessary perturbations to other parts of the design.

3 Our Approach

This section presents **CritiCell**, our framework for fine-grained timing attribution. Figure 2 provides an overview of CritiCell and its applications. Implementation details are provided in Appendix B.

3.1 Motivation

Timing optimization faces a **granularity mismatch between path-level feedback and cell-level optimization decisions**. STA identifies timing-critical paths through their slack values (Bhasker & Chadha, 2009). However, identifying a violating path does not directly determine which cells along that path should receive greater emphasis during timing optimization.

This mismatch matters because cells on the same critical path need not contribute equally to its unfavorable timing behavior. For example, a cell may exhibit unfavorable local delay or interconnect conditions, while other cells on the same path have less problematic local configurations. We refer to cells that are major contributors to timing degradation on a given path as **timing-critical cells**. Propagating path-level urgency without distinguishing these cells can distribute optimization effort too broadly. We therefore identify this granularity mismatch as a potential bottleneck to timing closure. Figure 1 illustrates the potential benefit of addressing this mismatch, using placement as a concrete example. Path-level guidance (top) distributes optimization effort across critical path C, potentially perturbing other paths such as A and B. In contrast, cell-level guidance (bottom) prioritizes timing-critical cells within C. This finer guidance can help target local bottlenecks while limiting unnecessary perturbations to other parts of the design.

This granularity mismatch raises a central question: how can we identify the cells that contribute most to timing degradation on a given path? The available path-level timing labels characterize the collective behavior of cells, without directly identifying these timing-critical cells. To bridge the gap between coarse-grained feedback and fine-grained optimization decisions, we develop CritiCell, a weakly supervised learning framework that learns cell-level timing importance from path-level supervision without requiring cell-level annotations. The resulting cell-level guidance complements STA and can be incorporated into existing timing-aware optimizers.

[1]Unless otherwise stated, *placement* refers to *global placement* in this paper.

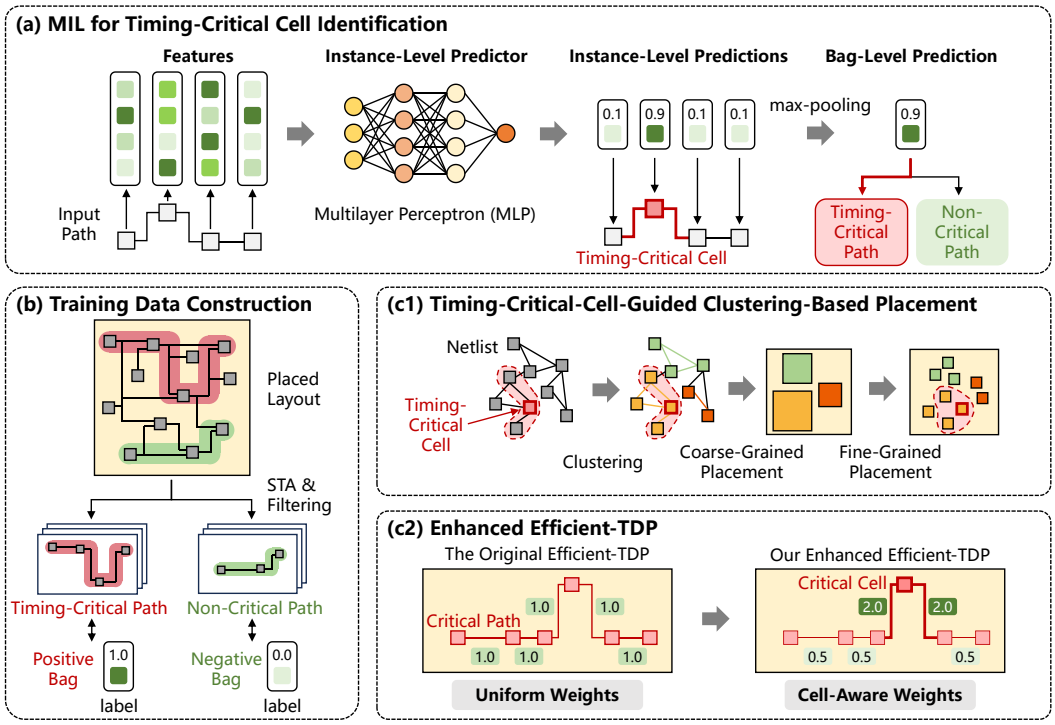


Figure 2: **Overview of CritiCell and its applications.** (a) We formulate timing-critical cell identification as a MIL problem, where each timing path is a *bag* and the cells on the path are *instances*; an instance-level predictor outputs per-cell criticality scores, and max-pooling aggregates them to predict the path label under weak (path-level) supervision. (b) We build the training set by running STA on placed layouts and filtering extracted paths to form positive and negative bags with path-level labels. (c1) The identified timing-critical cells provide fine-grained information to guide clustering-based placement. (c2) We further enhance Efficient-TDP by converting path-level pin-pair attraction weights into cell-aware weights.

3.2 Timing-Critical Cell Formulation

Following Section 3.1, we seek to identify timing-critical cells that are major contributors to timing degradation on a given path. Cell criticality is path-dependent: the same cell may be timing-critical on one path but non-critical on another. The identification target is therefore a cell in its path context.

For a timing path p, let $\mathcal{I}_p$ denote the index set of its cells and $y_p \in \{0, 1\}$ its STA-derived label, with $y_p = 1$ indicating a timing-critical path. We associate each cell $i \in \mathcal{I}_p$ with a latent label $z_{p,i} \in \{0, 1\}$, where $z_{p,i} = 1$ indicates that the cell is timing-critical with respect to p. Only the path labels y_p are observed during training; the cell labels $z_{p,i}$ are unavailable.

To connect these two levels of granularity, we adopt a multiple instance learning (MIL) abstraction: a timing-critical path is assumed to contain at least one timing-critical cell, whereas a non-critical path is assumed to contain none. Under this assumption,

$$y_p = \max_{i \in \mathcal{I}_p} z_{p,i}.$$

Each path thus forms a bag, and each cell on that path is treated as an instance. This relation serves as a modeling assumption for learning from path-level supervision. The path-filtering procedure described in Section 3.3 is designed to better align the training data with this abstraction.

Our objective is to learn an instance-level predictor that assigns timing-criticality scores to cells in their path context using only the observed bag labels. Aggregating these scores into a path-level prediction enables training with STA-derived supervision, while retaining cell-level predictions for timing-critical cell identification. Section 3.3 describes the feature representation, predictor, and training procedure.

3.3 CRITICELL FRAMEWORK

Building on the formulation in Section 3.2, we present the design of CritiCell, including its feature representation, model architecture, and training procedure for learning cell-level timing importance from path-level supervision.

Feature Design We use path-aware cell features to characterize local physical and timing conditions, together with the cell's context along the path. These features are grouped into four categories.

Geometric Placement Features. Placement quality has a direct impact on interconnect length and signal propagation distance, which in turn affects timing behavior. To capture the geometric context of a cell on a timing path, we include the spatial coordinates of the cell itself, the locations of its input and output pins, as well as the output pin location of the preceding cell and the input pin location of the succeeding cell along the path. These features provide localized geometric context that reflects how the cell is embedded within the physical realization of the timing path.

Timing Delay Features. To reflect the timing behavior reported by STA, we include delay values corresponding to the cell itself, as well as the immediately adjacent interconnect segments before and after the cell along the timing path.

Path Position Feature. We further include the relative position of the cell along the timing path, represented by its index within the path. This feature provides global path context, allowing the model to differentiate cells appearing at different stages of signal propagation, which may exhibit distinct timing sensitivities.

Local Interconnect Features. Finally, to characterize the local interconnect complexity surrounding the cell, we include the fanout counts of the nets immediately before and after the cell along the timing path. These features capture local connectivity and loading conditions, which can influence the timing behavior of the path.

Model Architecture Inspired by the mi-Net framework (Wang et al., 2018), we model timing-critical cell identification under the MIL formulation described in Section 3.2. Specifically, each cell on a timing path is treated as an instance. Let $\mathbf{x}_{p,i}$ denote the feature vector of the i-th cell on a given timing path p. An instance-level predictor, implemented as a multi-layer perceptron (MLP) with a sigmoid output, maps the cell features to a scalar value in $[0, 1]$:

$$s_{p,i} = f_{\text{MLP}}(\mathbf{x}_{p,i}), \tag{3}$$

where $s_{p,i}$ represents the timing-criticality score of the i-th cell with respect to path p.

To aggregate instance-level predictions into a path-level prediction, we employ a max-pooling operation over all cells on the path:

$$\hat{y}_p = \max_{i \in \mathcal{I}_p} s_{p,i}, \tag{4}$$

where $\mathcal{I}_p$ denotes the index set of cells along timing path p. This aggregation is well aligned with our modeling assumption that a timing-critical path contains at least one timing-critical cell, whereas a non-critical path contains none.

The pooled output $\hat{y}_p$ is used to perform timing-critical versus non-critical path classification, and can be directly supervised using path-level labels derived from STA.

Model Training To train the model, we construct a dataset $\mathcal{D}$ using a collection of C circuits. For each circuit, we generate a set of M diverse layouts $\{X_{c,m}\}_{m=1}^{M}$.

For each generated layout $X_{c,m}$, we apply STA to extract a large number of timing-critical and non-critical timing paths. As discussed in Section 3.2, our MIL formulation relies on a path-level abstraction that associates timing-critical paths with the presence of at least one timing-critical cell, while non-critical paths are assumed to contain no such cells. While, for practical modeling reasons, this assumption involves a certain simplification and may not strictly hold in all cases, we seek to better align the training data with this abstraction through the following filtering steps.

Specifically, we apply two filters sequentially to the extracted timing paths. First, we retain only the n timing-critical paths with the smallest slack values and the n non-critical paths with the largest

slack values. This step ensures that positive bags correspond to paths with clear timing violations, while negative bags correspond to paths that comfortably meet timing constraints.

Second, we further filter paths based on their length in terms of the number of cells. We retain timing-critical paths with fewer than L cells and non-critical paths with more than l cells. This criterion aims to exclude timing violations that are primarily caused by excessive path length, rather than poor local design choices, and to avoid trivially short paths that may meet timing constraints despite unfavorable local design conditions.

After filtering, the resulting dataset is constructed as

$$\mathcal{D} = \bigcup_{c\in[C],\, m\in[M]} \left(\mathcal{P}^{+}_{c,m} \cup \mathcal{P}^{-}_{c,m}\right), \tag{5}$$

where $\mathcal{P}^{+}_{c,m}$ and $\mathcal{P}^{-}_{c,m}$ denote the sets of filtered timing-critical and non-critical paths extracted from layout $X_{c,m}$, respectively. Each path $p \in \mathcal{D}$ corresponds to a bag with label $y_p \in \{0, 1\}$, where $y_p = 1$ for $p \in \mathcal{P}^{+}_{c,m}$ and $y_p = 0$ otherwise. We train the model using the binary cross-entropy loss.

3.4 Applications

We demonstrate how the fine-grained guidance learned by CritiCell supports two timing-aware placement applications: *timing-critical-cell-guided clustering-based placement* and an *enhanced Efficient-TDP*. These two applications serve as representative instantiations of using timing-critical cells for constraint-driven and objective-driven timing optimization, respectively.

Timing-Critical-Cell-Guided Clustering-Based Placement We follow the idea of clustering-based placement (Fogaca et al., 2020; Lu et al., 2021; 2022; Kahng et al., 2024) and incorporate the identified timing-critical cells to impose explicit, constraint-driven guidance for timing optimization.

Specifically, in the initial clustering stage, we introduce a soft grouping bias that encourages timing-critical cells on a timing-critical path, together with their neighboring cells along the same path, to be grouped into the same cluster. By focusing clustering decisions on cells identified as dominant contributors to timing degradation, the resulting clusters more accurately capture the true timing bottlenecks along critical paths.

After clustering, a coarse-grained placement is performed on the clustered netlist, treating each cluster as a single block. In the subsequent fine-grained placement stage, the coarse-grained placement solution is translated into explicit spatial constraints by initializing all cells within a cluster at the geometric center of the corresponding cluster location. Starting from this initialization, an incremental placement is performed, during which nets connecting timing-critical cells and their neighboring cells along the same timing-critical path are assigned larger weights. These constraints explicitly emphasize the placement of timing-critical cells, encouraging closer proximity among the dominant timing bottlenecks, and leading to improved timing quality.

Enhanced Efficient-TDP with Timing-Critical Cells We further apply the proposed timing-critical cell identification framework to enhance Efficient-TDP (Shi et al., 2025), a representative objective-driven timing optimization method. In Efficient-TDP, all pin pairs along a timing-critical path are treated uniformly in the pin-to-pin attraction term. However, this path-level treatment does not distinguish cells that are true timing bottlenecks from those that are less critical.

To address this limitation, we introduce a cell-aware reweighting mechanism. Specifically, we augment the original pin-pair weight update rule in Efficient-TDP by introducing a modulation factor α_{ij}:

$$w(i,j) = \begin{cases} w_0, & (i,j) \notin P, \\ w(i,j) + \alpha_{ij} \cdot w_1 \cdot (\text{slack}/\text{WNS}), & \text{otherwise}, \end{cases} \tag{6}$$

where (i, j) denotes a pin pair on a timing-critical path. The factor $\alpha_{i,j}$ is determined based on the timing-criticality of the cells associated with the pin pair. Pin pairs associated with at least one timing-critical cell are assigned larger $\alpha_{i,j}$ values, while the rest receive smaller values.

By differentiating the importance of pin pairs at the cell level, the modified objective places stronger emphasis on shortening interconnects associated with true timing bottlenecks. This enables more targeted timing optimization compared to the original Efficient-TDP formulation.

Table 1: **Comparison of WNS ($\times 10^3$ ps) and TNS ($\times 10^5$ ps) on `ICCAD2015` benchmarks under the clustering-based placement setting.** Best values are shown in **bold**. The Improvement row reports relative gains over DREAMPlace.

	DREAMPlace		TritonPart		Ours (Clust.)	
	WNS	TNS	WNS	TNS	WNS	TNS
superblue1	-22.33	-287.87	–†		**-19.85**	**-225.27**
superblue3	-33.14	-160.93	-29.80	-180.53	**-23.37**	**-138.27**
superblue4	-22.62	-207.46	-22.23	-206.27	**-21.77**	**-203.41**
superblue5	**-45.14**	**-199.01**	-47.04	-265.07	-46.49	-234.33
superblue7	-19.73	-226.80	**-16.81**	-226.82	-19.20	**-220.89**
superblue10	-27.17	**-896.78**	-27.46	-899.82	**-25.42**	-902.71
superblue16	-19.23	-302.44	**-14.96**	-273.19	-15.29	**-234.60**
superblue18	-17.78	-111.19	-12.18	-96.24	**-11.66**	**-74.70**
Improvement	Ref.		10.72%	-3.15%	**13.17%**	**9.65%**

† TritonPart on superblue1 yields an extreme timing outlier and is excluded from the Improvement summary. Compared directly with TritonPart, Ours achieves **2.87%/10.74%** average WNS/TNS improvements on the 7 valid designs.

4 Experiments

4.1 Experiment Setup

Benchmark We evaluate on the ICCAD2015 timing-driven placement benchmark suite, which contains eight large-scale industrial circuits with up to over one million standard cells.

Baselines For the proposed *timing-critical-cell-guided clustering-based placement*, we consider two baselines. We use vanilla DREAMPlace (Lin et al., 2019) as a timing-unaware baseline, as it performs global placement without incorporating timing information. We also include a path-level timing-aware baseline by replacing the clustering component in our framework with TritonPart (Bustany et al., 2023), a timing-aware clustering method that leverages path-level timing information. For the proposed *enhanced Efficient-TDP with timing-critical cells*, we compare against the original Efficient-TDP (Shi et al., 2025).

Evaluation Metrics We evaluate timing performance using WNS and TNS, as introduced in Section 2.3. For the *timing-critical-cell-guided clustering-based placement* and its corresponding baselines, we evaluate timing using OpenSTA. For the *enhanced Efficient-TDP with timing-critical cells* and the original Efficient-TDP, we report the timing metrics obtained from the timing analysis module integrated in the Efficient-TDP framework.

Training Details The timing-critical cell identification model is trained on a subset of circuits from `ICCAD2015` and evaluated on the full benchmark suite. Specifically, we use three circuits for training and two circuits for validation, while the remaining three circuits are reserved for *zero-shot* testing. Following the dataset construction procedure described in Section 3.3, we generate $M{=}3$ diverse layouts using DREAMPlace for each training/validation circuit, from which timing-critical and non-critical paths are extracted using OpenSTA (OpenSTA, 2023). We train the model with a learning rate of 10^{-3} for up to 1000 epochs, and select the checkpoint with the lowest validation loss as the final model.

4.2 Main Results

Table 1 reports the WNS and TNS on `ICCAD2015` under the clustering-based placement setting. The results show that timing-critical-cell-guided constraints generally improve timing quality across designs. Compared with vanilla DREAMPlace, our timing-critical-cell-guided clustering-based placement achieves average improvements of **13.17%** in WNS and **9.65%** in TNS, and further

Table 2: **Comparison of WNS ($\times 10^3$ ps) and TNS ($\times 10^5$ ps) on `ICCAD2015` benchmarks under the Efficient-TDP-based placement setting.** Best values are shown in **bold**. The Improvement row reports relative gains over Efficient-TDP.

	Efficient-TDP		Ours (TDP)	
	WNS	TNS	WNS	TNS
superblue1	-7.75	-17.44	**-7.22**	**-14.49**
superblue3	-11.82	-20.40	**-11.31**	**-18.93**
superblue4	-9.17	-82.88	**-8.69**	**-81.02**
superblue5	**-24.65**	-62.18	-24.84	**-60.35**
superblue7	**-15.22**	-43.52	-15.40	**-31.33**
superblue10	-23.08	-558.14	**-22.97**	**-555.75**
superblue16	-8.63	-22.90	**-8.02**	**-19.22**
superblue18	-6.92	-16.16	**-6.18**	**-14.44**
Improvement	Ref.		**4.08%**	**10.56%**

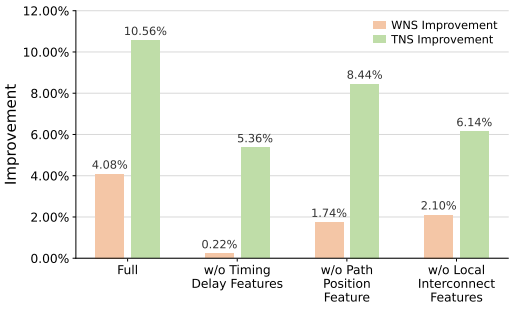


Figure 3: Feature ablation study on `ICCAD2015`. Average relative WNS and TNS improvements over the original Efficient-TDP when integrating identified timing-critical cells, comparing the full feature set with variants that remove one feature group at a time.

achieves average WNS/TNS improvements of **2.87%/10.74%** over the TritonPart-based path-level timing-aware baseline. Table 2 summarizes the results under the Efficient-TDP-based setting. By enhancing Efficient-TDP with timing-critical cells, our method delivers additional gains, with average improvements of **4.08%** in WNS and **10.56%** in TNS over Efficient-TDP.

Overall, these results demonstrate that the identified timing-critical cells provide effective fine-grained guidance for timing-aware design across different integration paradigms. Additional experimental results and placement visualizations are provided in Appendices C and D.

4.3 Feature Ablation Study

We ablate feature groups in our timing-critical cell identification model to assess their contributions. In our placement setting, we regard *Geometric Placement Features* as basic features describing local cell placement. We therefore keep these features always on and remove one of *Timing Delay Features*, *Path Position Feature*, or *Local Interconnect Features* at a time, retraining the MIL model under the same protocol. We then plug the predicted cell criticality into our enhanced Efficient-TDP pipeline and report average WNS/TNS improvements over the original Efficient-TDP baseline on `ICCAD2015` (Fig. 3).

The results indicate that **Timing Delay Features are particularly important**: removing them leads to a substantial drop in timing improvement, decreasing the gains to **0.22%** in WNS and **5.36%** in TNS. This observation aligns well with intuition, as delay-related information directly reflects the timing behaviors revealed by STA. Meanwhile, ablating either *Path Position Feature* or *Local Interconnect Features* also consistently degrades performance, suggesting that these features provide complementary information for learning effective cell-level timing guidance.

5 Conclusion

We presented CritiCell, a weakly supervised learning framework that bridges path-level timing feedback and cell-level optimization guidance. Through a multiple instance learning formulation, CritiCell learns cell-level timing importance from STA-derived path labels without requiring cell-level annotations. Experiments on `ICCAD2015` demonstrate that the learned guidance improves WNS and TNS in both constraint-driven and objective-driven placement. These results highlight the potential of learning fine-grained optimization guidance from aggregated observations, offering a way to address the granularity mismatch between available feedback and optimization decisions.

AI USE STATEMENT

In this work, we did not use generative AI tools for any tasks with required disclosure. We have not used generative AI tools for the following tasks: (i) generate synthetic data sets; (ii) help develop theoretical models or conceptual frameworks; (iii) formulate mathematical claims; (iv) propose or refine hypotheses; (v) design or provide feedback on research methodology or experiments; (vi) implement methods; (vii) assist with translation; (viii) clean and reformat dataset; (ix) support qualitative and thematic data analysis; (x) interpret results. The tasks "provide critical ingredients for proving mathematical claims" and "assist in the writing of proofs" are not applicable to this work. Additionally, we used generative AI tools for the task "edit a research paper to improve readability", solely to polish the grammar, clarity, and readability of the manuscript. We have reviewed all AI-assisted work. Specifically, the authors checked the revised text for consistency with the intended technical meaning and revised it as needed. We take responsibility for the final content of this work, including text, claims or artifacts produced with the aid of generative AI.

REFERENCES

Jayaram Bhasker and Rakesh Chadha. *Static timing analysis for nanometer designs: A practical approach*. Springer Science & Business Media, 2009.

Ismail Bustany, Grigor Gasparyan, Andrew B Kahng, Ioannis Koutis, Bodhisatta Pramanik, and Zhiang Wang. An open-source constraints-driven general partitioning multi-tool for vlsi physical design. In *2023 IEEE/ACM International Conference on Computer Aided Design (ICCAD)*, pp. 1–9. IEEE, 2023.

H Chang, Eugene Shragowitz, Jian Liu, Habib Youssef, Bing Lu, and Suphachai Sutanthavibul. Net criticality revisited: An effective method to improve timing in physical design. In *Proceedings of the 2002 international symposium on Physical design*, pp. 155–160, 2002.

Lei Chen, Yiqi Chen, Zhufei Chu, Wenji Fang, Tsung-Yi Ho, Yu Huang, Sadaf Khan, Min Li, Xingquan Li, Yun Liang, et al. The dawn of ai-native eda: Promises and challenges of large circuit models. *arXiv preprint arXiv:2403.07257*, 2024.

Mateus Fogaca, Andrew B Kahng, Eder Monteiro, Ricardo Reis, Lutong Wang, and Mingyu Woo. On the superiority of modularity-based clustering for determining placement-relevant clusters. *Integration*, 74:32–44, 2020.

Bangqi Fu, Lixin Liu, Martin DF Wong, and Evangeline FY Young. Hybrid modeling and weighting for timing-driven placement with efficient calibration. In *Proceedings of the 43rd IEEE/ACM International Conference on Computer-Aided Design*, pp. 1–9, 2024.

Jiaqi Gu, Zixuan Jiang, Yibo Lin, and David Z Pan. Dreamplace 3.0: Multi-electrostatics based robust vlsi placement with region constraints. In *Proceedings of the 39th International Conference on Computer-Aided Design*, pp. 1–9, 2020.

Zizheng Guo and Yibo Lin. Differentiable-timing-driven global placement. In *Proceedings of the 59th ACM/IEEE Design Automation Conference*, pp. 1315–1320, 2022.

Zizheng Guo, Mingjie Liu, Jiaqi Gu, Shuhan Zhang, David Z Pan, and Yibo Lin. A timing engine inspired graph neural network model for pre-routing slack prediction. In *Proceedings of the 59th ACM/IEEE Design Automation Conference*, pp. 1207–1212, 2022.

Andrew Kahng, Seokhyeong Kang, Sayak Kundu, Kyungjun Min, Seonghyeon Park, and Bodhisatta Pramanik. Ppa-relevant clustering-driven placement for large-scale vlsi designs. In *Proceedings of the 61st ACM/IEEE Design Automation Conference*, pp. 1–6, 2024.

Yao Lai, Sungyoung Lee, Guojin Chen, Souradip Poddar, Mengkang Hu, David Z Pan, and Ping Luo. Analogcoder: Analog circuit design via training-free code generation. In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 39, pp. 379–387, 2025.

Peiyu Liao, Siting Liu, Zhitang Chen, Wenlong Lv, Yibo Lin, and Bei 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.

Jai Ming Lin, You Yu Chang, and Wei Lun Huang. Timing-driven analytical placement according to expected cell distribution range. In *Proceedings of the 2024 International Symposium on Physical Design (ISPD 2024)*, pp. 177–184, Taipei, Taiwan, 2024. Association for Computing Machinery. doi: 10.1145/3626184.3633318.

Yibo Lin, Shounak Dhar, Wuxi Li, Haoxing Ren, Brucek Khailany, and David Z Pan. Dreamplace: Deep learning toolkit-enabled gpu acceleration for modern vlsi placement. In *Proceedings of the 56th Annual Design Automation Conference 2019*, pp. 1–6, 2019.

Yibo Lin, David Z Pan, Haoxing Ren, and Brucek Khailany. Dreamplace 2.0: Open-source gpu-accelerated global and detailed placement for large-scale vlsi designs. In *2020 China Semiconductor Technology International Conference (CSTIC)*, pp. 1–4. IEEE, 2020.

Yi-Chen Lu, Sai Pentapati, and Sung Kyu Lim. The law of attraction: Affinity-aware placement optimization using graph neural networks. In *Proceedings of the 2021 International Symposium on Physical Design*, pp. 7–14, 2021.

Yi-Chen Lu, Tian Yang, Sung Kyu Lim, and Haoxing Ren. Placement optimization via ppa-directed graph clustering. In *Proceedings of the 2022 ACM/IEEE Workshop on Machine Learning for CAD*, pp. 1–6, 2022.

Don MacMillen, Raul Camposano, D Hill, and Thomas W Williams. An industrial view of electronic design automation. *IEEE transactions on computer-aided design of integrated circuits and systems*, 19(12):1428–1448, 2000.

Igor L Markov, Jin Hu, and Myung-Chul Kim. Progress and challenges in vlsi placement research. In *Proceedings of the International Conference on Computer-Aided Design*, pp. 275–282, 2012.

OpenSTA. `https://github.com/The-OpenROAD-Project/OpenSTA`, 2023.

Jan M Rabaey, Anantha P Chandrakasan, and Borivoje Nikolic. *Digital integrated circuits*, volume 2. Prentice hall Englewood Cliffs, 2002.

Yunqi Shi, Siyuan Xu, Shixiong Kai, Xi Lin, Ke Xue, Mingxuan Yuan, and Chao Qian. Timing-driven global placement by efficient critical path extraction. *arXiv preprint arXiv:2503.11674*, 2025.

William Swartz and Carl Sechen. Timing driven placement for large standard cell circuits. In *Proceedings of the 32nd Annual ACM/IEEE Design Automation Conference*, pp. 211–215, 1995.

Shailja Thakur, Baleegh Ahmad, Hammond Pearce, Benjamin Tan, Brendan Dolan-Gavitt, Ramesh Karri, and Siddharth Garg. Verigen: A large language model for verilog code generation. *ACM Transactions on Design Automation of Electronic Systems*, 29(3):1–31, 2024.

Vincent A Traag, Ludo Waltman, and Nees Jan Van Eck. From louvain to leiden: guaranteeing well-connected communities. *Scientific reports*, 9(1):1–12, 2019.

Laung-Terng Wang, Yao-Wen Chang, and Kwang-Ting Tim Cheng. *Electronic design automation: synthesis, verification, and test*. Morgan Kaufmann, 2009.

Xinggang Wang, Yongluan Yan, Peng Tang, Xiang Bai, and Wenyu Liu. Revisiting multiple instance neural networks. *Pattern recognition*, 74:15–24, 2018.

Zhihai Wang, Lei Chen, Jie Wang, Yinqi Bai, Xing Li, Xijun Li, Mingxuan Yuan, Jianye Hao, Yongdong Zhang, and Feng Wu. A circuit domain generalization framework for efficient logic synthesis in chip design. In *Forty-first International Conference on Machine Learning*. PMLR, 2024a.

Zhihai Wang, Jie Wang, Dongsheng Zuo, Yunjie Ji, Xinli Xia, Yuzhe Ma, Jianye Hao, Mingxuan Yuan, Yongdong Zhang, and Feng Wu. A hierarchical adaptive multi-task reinforcement learning framework for multiplier circuit design. In *Forty-first International Conference on Machine Learning*. PMLR, 2024b.

Muhammad Waqas, Syed Umaid Ahmed, Muhammad Atif Tahir, Jia Wu, and Rizwan Qureshi. Exploring multiple instance learning (mil): A brief survey. *Expert Systems with Applications*, 250: 123893, 2024.

Yuyang Ye, Tinghuan Chen, Yifei Gao, Hao Yan, Bei Yu, and Longxing Shi. Fast and accurate wire timing estimation based on graph learning. In *2023 Design, Automation & Test in Europe Conference & Exhibition (DATE)*, pp. 1–6. IEEE, 2023a.

Yuyang Ye, Tinghuan Chen, Yifei Gao, Hao Yan, Bei Yu, and Longxing Shi. Graph-learning-driven path-based timing analysis results predictor from graph-based timing analysis. In *Proceedings of the 28th Asia and South Pacific Design Automation Conference*, pp. 547–552, 2023b.

Haisheng Zheng, Zhuolun He, Fangzhou Liu, Zehua Pei, and Bei Yu. Lstp: A logic synthesis timing predictor. In *2024 29th Asia and South Pacific Design Automation Conference (ASP-DAC)*, pp. 728–733. IEEE, 2024.

A SUPPLEMENTAL BACKGROUND

A.1 BASIC CONCEPTS IN CHIP DESIGN

A digital chip consists of a collection of circuit components connected by electrical interconnects. At a high level, these components can be broadly categorized into *cells*, *macros*, and *I/O elements*.

Cells (also referred to as standard cells) are the fundamental building blocks of digital logic, including logic gates (e.g., AND, OR, XOR), buffers, and flip-flops. Cells are typically small, numerous, and freely movable during placement.

Macros are large, complex, and pre-designed blocks such as SRAMs, analog IPs, or specialized accelerators. Macros usually have fixed internal layouts and are treated as rigid blocks during physical design. In many practical design flows, macro placement is either fixed beforehand or handled by specialized algorithms.

In this work, we do not consider macro placement. We assume that macro locations are given and fixed, and our placement optimization focuses exclusively on standard cells.

A chip also interacts with its environment through **primary inputs (PIs)** and **primary outputs (POs)**, which represent external signal sources and sinks, respectively. Signals propagate from PIs or the outputs of sequential elements (e.g., flip-flops), traverse combinational logic composed of cells and nets, and eventually arrive at POs or the inputs of sequential elements.

A.2 STATIC TIMING ANALYSIS

Static Timing Analysis (STA) is a standard and widely adopted technique for evaluating the timing quality of a chip without relying on dynamic simulation (Bhasker & Chadha, 2009). Given a circuit and a target clock period, STA analyzes signal propagation delays (i.e., the time a signal takes to travel from one pin to another) and identifies potential timing violations.

Formally, STA models a digital chip as a directed graph $G = (V, E)$, where each node $v \in V$ corresponds to a pin, and each directed edge $e \in E$ represents either a cell-internal connection or a net connection between pins.

A central concept in STA is the **timing path**, which serves as the fundamental unit of timing analysis. Intuitively, a timing path represents a sequence of pins through which a signal propagates within a single clock cycle. Formally, a timing path

$$p = (v_1, v_2, \ldots, v_n)$$

is an ordered sequence of pins $v_i \in V$, where each consecutive pair $(v_i, v_{i+1}) \in E$. The path starts at v_1, which corresponds to either the output pin of a flip-flop or a primary input, and ends at v_n, which corresponds to either the input pin of a flip-flop or a primary output.

For a timing path, the **arrival time (AT)** is defined as the total propagation delay accumulated along the path. Each timing path is also associated with a **required arrival time (RAT)**, which specifies the latest allowable time for the signal to arrive without violating timing constraints imposed by the clock.

Based on the arrival time and the required arrival time, STA evaluates timing quality using the **slack**, defined as

$$\mathrm{Slack}(p) = \mathrm{RAT}(p) - \mathrm{AT}(p).$$

A path with negative slack violates the timing constraint and is referred to as a *timing-critical path*, with more negative slack indicating more severe timing violations.

Two standard chip-level timing metrics derived from slack are commonly reported. The **worst negative slack (WNS)** is the minimum slack among all timing paths and reflects the severity of the most critical timing violation. The **total negative slack (TNS)** is the sum of negative slack values across all violating timing paths and captures the overall extent of timing violations in the design. For both metrics, values closer to zero indicate better timing performance.

A.3 Cell Placement

Cell placement is a key stage in chip physical design, aiming to determine the physical locations of cells on the chip canvas. In practice, cell placement is typically divided into two phases: **global placement (GP)** and **detailed placement (DP)**. Global placement computes approximate cell locations in a continuous layout space, while detailed placement legalizes these locations onto discrete sites and removes overlaps.

From an optimization perspective, global placement is commonly formulated as a continuous optimization problem (Lin et al., 2019; 2020; Gu et al., 2020; Liao et al., 2022). Let $(\mathbf{x}, \mathbf{y})$ denote the coordinates of all cells on the chip plane, and let $\mathcal{N}$ denote the set of nets. A widely used formulation is

$$\min_{(\mathbf{x},\mathbf{y})} \sum_{n\in\mathcal{N}} W_n(\mathbf{x},\mathbf{y}) + \lambda \cdot D(\mathbf{x},\mathbf{y}), \tag{7}$$

where $W_n(\cdot)$ represents a smoothed approximation of half-perimeter wirelength (HPWL) for net n, and $D(\cdot)$ penalizes excessive overlap and density to encourage well-distributed cell placement. The scalar λ balances wirelength optimization and density control.

This continuous and differentiable formulation enables the application of gradient-based optimization methods to directly update cell coordinates. DREAMPlace (Lin et al., 2019) is a representative open-source global placement framework that accelerates large-scale placement optimization using GPUs.

A.4 Machine Learning for EDA

Recently, machine learning (ML) techniques have demonstrated strong potential across multiple stages of the EDA pipeline (Chen et al., 2024), including RTL code generation (Thakur et al., 2024), logic synthesis (Wang et al., 2024a), and circuit design (Wang et al., 2024b; Lai et al., 2025). Prior work has also explored learning-based approaches for timing-related tasks, such as timing prediction (Guo et al., 2022; Ye et al., 2023b; Zheng et al., 2024; Ye et al., 2023a). These advances suggest an opportunity to use ML to extract richer information from existing EDA signals, potentially overcoming limitations inherent in traditional analysis tools.

B Implementation Details

B.1 Model Details

To improve numerical stability during training and to facilitate learning across different timing paths and chips, we apply feature normalization to several input features. Specifically, spatial coordinate features are normalized to the range $[0, 1]$ based on the chip dimensions. Delay-related features are normalized by dividing by the required arrival time of the corresponding timing path. In addition, the path position index is normalized by the total number of cells on the path, yielding a relative position along the path.

The instance-level predictor is implemented as a multi-layer perceptron (MLP) that takes a 16-dimensional feature vector as input. The MLP consists of two hidden layers with 32 units each, followed by a single-unit output layer. ReLU is used as the activation function for the hidden layers, and a sigmoid activation is applied at the output layer to produce a timing-criticality score in the range $[0, 1]$. In both placement applications, a cell is identified as timing-critical when its predicted criticality score is at least 0.9.

B.2 Training Environment and Runtime

All experiments were conducted on a workstation equipped with an Intel Xeon E5-2667 v4 CPU and an NVIDIA GeForce RTX 2080 Ti GPU. The total training time for one run is 101.07 seconds.

B.3 Dataset Details

Circuit statistics Table 3 summarizes the size statistics of the eight circuits from the `ICCAD2015` benchmark suite used in this work.

Table 3: Statistics of eight circuits from the `ICCAD2015` benchmark suite.

Circuit	#Macros	#Standard Cells	#Nets	#Pins
`superblue1`	424	1215820	1215710	3767494
`superblue3`	565	1219170	1224979	3905321
`superblue4`	300	801968	802513	2497940
`superblue5`	770	1090247	1100825	3246878
`superblue7`	441	1937699	1933945	6372094
`superblue10`	1629	984379	1898119	5560506
`superblue16`	99	985909	999902	3013268
`superblue18`	201	771845	771542	2559143

Dataset split We split circuits in `ICCAD2015` into training/validation/unseen sets. The training set includes `superblue1`, `superblue3`, and `superblue4`; the validation set includes `superblue5` and `superblue7`; the unseen test set includes `superblue10`, `superblue16`, and `superblue18`.

Layout generation For each training/validation circuit, we generate three diverse layouts. Specifically, we first run DREAMPlace to obtain one layout. For the remaining layouts, we randomly select a subset of cells and randomly fix their locations before running DREAMPlace.

Path-length filtering We set $L = 70$ and $l = 30$ for path-length filtering, with both thresholds inclusive.

B.4 Timing-Critical-Cell-Guided Clustering-Based Placement

We describe the implementation of our timing-critical-cell-guided clustering-based *placement* pipeline. The pipeline consists of two placement stages: a *coarse-grained placement* that places clusters (implemented as fake macros), followed by a *fine-grained cell placement* initialized from the coarse solution and guided by timing-informed weights.

Stage 0: Initial placement and critical-path extraction We first run DREAMPlace on the original cell-level netlist to obtain an initial placement. We then perform STA on this placement and extract the top-K most timing-critical paths.

Timing-informed groups from timing-critical cells For each extracted timing-critical path, we apply our timing-critical cell identification model to obtain a set of timing-critical cells on the path. Each identified timing-critical cell is paired with its adjacent cell(s) along the same path to form a small group. Taking the union of all such groups over the top-K paths and merging overlaps yields a set of disjoint groups, where each group contains cells that are encouraged to be co-located.

Clustering via a timing-weighted cell graph We build an undirected graph whose vertices correspond to cells. An edge is added between two cells if they are connected by at least one net. To promote grouping consistency in clustering, we assign a significantly larger weight to edges whose endpoints fall in the same timing-informed group, while keeping other edges at the default weight. We then apply the Leiden algorithm (Traag et al., 2019) on this weighted graph to obtain a partition of cells into clusters.

Stage 1: Coarse-grained placement with fake macros To enable coarse-grained placement, we represent each cluster as a *fake macro* whose area equals the total area of cells in that cluster, producing a cluster-level instance set. We run DREAMPlace to place these fake macros, yielding a coarse placement solution at the cluster granularity.

Stage 2: Fine-grained cell placement initialized from clusters We then restore the original cells by mapping each fake macro back to its member cells and initializing these cells at the geometric center of the corresponding fake macro. Starting from this initialization, we run DREAMPlace again to obtain the final cell-level placement. During this fine-grained placement stage, we apply timing-informed net reweighting: if a net connects at least two cells that belong to the same timing-informed group, we assign this net a larger weight than ordinary nets to further encourage local co-location.

Top-K setting Table 4 summarizes the value of K used for extracting top-K timing-critical paths on each `ICCAD15` circuit.

Table 4: Values of K for top-K timing-critical path extraction in the clustering-based placement setting.

Design	K
superblue1	1000
superblue3	3000
superblue4	1000
superblue5	1000
superblue7	3000
superblue10	2000
superblue16	3000
superblue18	1000

B.5 Enhanced Efficient-TDP with Timing-Critical Cells

We detail the implementation of *Enhanced Efficient-TDP* that incorporates timing-critical cells to modulate the pin-pair attraction weights. The method alternates between (i) running STA to extract timing-critical paths, (ii) identifying timing-critical cells on these paths, (iii) updating pin-pair weights accordingly, and (iv) continuing gradient-based global placement with the augmented objective.

Algorithm 1 summarizes the implementation of Enhanced Efficient-TDP with timing-critical cells.

Algorithm 1. Enhanced Efficient-TDP with timing-critical cells for cell-aware pin-pair reweighting.

Input: placement variables $(\mathbf{x}, \mathbf{y})$; T_0, $T_{\max}$, ΔT; w_0, w_1; $\alpha_{\text{non}}, \alpha_{\text{crit}}$ with $\alpha_{\text{non}} < \alpha_{\text{crit}}$; coefficients λ, β.
Output: final placement $(\mathbf{x}, \mathbf{y})$.
Warm-up (wirelength + density only).
for $t = 1$ **to** T_0 **do**
  Take one GD step on $\sum_{n \in \mathcal{N}} W_n(\mathbf{x}, \mathbf{y}) + \lambda D(\mathbf{x}, \mathbf{y})$.
  if converged **then**
    return $(\mathbf{x}, \mathbf{y})$.
  end if
end for
$P \leftarrow \emptyset$.
Initialize $w(i, j)$ as undefined for all pin pairs.
while total GD steps $\leq T_{\max}$ **do**
  (1) Timing feedback.
  `Paths` $\leftarrow$ extract timing-critical paths by STA on current placement.
  WNS $\leftarrow$ worst negative slack reported by STA.
  (2) Cell-aware weight update.
  for each $p \in$ `Paths` **do**
    `Cells` $\leftarrow$ identify timing-critical cells on path p.
    $s \leftarrow \frac{|\text{slack}(p)|}{|\text{WNS}|}$.
    for each pin pair (i, j) on path p **do**
      if $(i, j) \notin P$ **then**
        $P \leftarrow P \cup \{(i, j)\}$; $w(i, j) \leftarrow w_0$.
      else
        if i or j belongs to a cell in `Cells` **then**
          $w(i, j) \leftarrow w(i, j) + \alpha_{\text{crit}} \cdot w_1 \cdot s$.

```
            else
              w(i,j) ← w(i,j) + α_non · w_1 · s.
            end if
          end if
        end for
      end for
      (3) Placement with augmented objective.
      PP(x, y) ← Σ_{(i,j)∈P} w(i,j) Q(i,j).
      for τ = 1 to ΔT do
        Take one GD step on Σ_{n∈N} W_n(x, y) + λD(x, y) + β PP(x, y).
        if converged then
          return (x, y).
        end if
      end for
    end while
    return (x, y).
```

Table 5 reports the values of α_{non} and α_{crit} used in our experiments on `ICCAD2015`. Unless otherwise specified, all other hyperparameters follow the original Efficient-TDP implementation.

Table 5: Values of α_{non} and α_{crit} used in the Enhanced Efficient-TDP setting on `ICCAD2015`.

Design	α_{non}	α_{crit}
superblue1	0.8	2.0
superblue3	0.0	1.5
superblue4	0.7	2.0
superblue5	0.8	2.0
superblue7	0.5	2.0
superblue10	0.8	2.0
superblue16	0.3	1.2
superblue18	0.0	2.0

C Additional Experiments and Analysis

C.1 Additional HPWL Results

For completeness, we list the half-perimeter wirelength (HPWL) values of the final layouts for reference. Table 6 lists the HPWL values under the clustering-based placement setting, while Table 7 lists those under the Efficient-TDP-based placement setting. All HPWL values are reported in units of 10^6.

C.2 Runtime under the Efficient-TDP-Based Placement Setting

We further report the placement runtime under the Efficient-TDP-based placement setting. Table 8 summarizes the runtime extracted from the final line of each DREAMPlace placement log. Overall, incorporating timing-critical-cell guidance introduces only limited additional placement time. We do not report runtime under the clustering-based placement setting, since TritonPart clustering takes substantially longer than our clustering procedure.

C.3 Correlation Between Local Delay and MIL Scores

We examine whether the learned MIL scores simply reproduce the local-delay ranking. For a cell c on path p, we define its *local delay* as

$$d_{\text{local}}(c,p) = d_{\text{arc}}(c,p) + d_{\text{net,prev}}(c,p) + d_{\text{net,next}}(c,p), \tag{8}$$

where d_{arc} is the cell-internal arc delay, and $d_{\text{net,prev}}$ and $d_{\text{net,next}}$ are the delays of the immediately preceding and following net segments along the path, respectively.

Table 6: **Reported HPWL values ($\times 10^6$) under the clustering-based placement setting.**

Design	DREAMPlace	TritonPart	Ours (Clust.)
superblue1	422.00	–	416.00
superblue3	478.20	464.83	463.85
superblue4	312.00	318.69	319.96
superblue5	488.30	479.95	478.28
superblue7	604.30	605.53	605.39
superblue10	935.90	902.74	909.71
superblue16	435.80	429.69	429.33
superblue18	243.00	236.12	236.83

Table 7: **Reported HPWL values ($\times 10^6$) under the Efficient-TDP-based placement setting.**

Design	Efficient-TDP	Ours (TDP)
superblue1	418.80	426.38
superblue3	462.50	470.85
superblue4	317.70	323.41
superblue5	484.20	490.87
superblue7	597.50	607.79
superblue10	911.60	925.27
superblue16	471.60	468.33
superblue18	234.40	238.97

For each `ICCAD2015` design, we extract 3000 timing-critical paths and compute the Spearman correlation between per-cell local delay and the MIL-predicted criticality score on each path. Figure 4 shows the distribution of per-path Spearman correlations.

The histogram is concentrated in the positive range, with mean 0.335 and median 0.316. Most paths exhibit weak-to-moderate positive correlation, while only a small fraction show negative correlation. This indicates that MIL predictions are related to local-delay score, but do not collapse to a simple local-delay ranking. Instead, the learned model appears to use local delay as one important cue while also incorporating additional placement, path-context, and interconnect information.

C.4 Comparison with Non-ML Cell-Criticality Heuristics

We further compare our MIL-based timing-critical cell identification method with simple non-ML cell-criticality heuristics. This experiment is designed to examine whether manually designed local timing and placement cues are sufficient to replace the learned cell-level attribution model.

Local-delay heuristic. The first baseline ranks cells on each timing-critical path by their local-delay score. For a cell c on path p, we use the local delay $d_{\text{local}}(c, p)$ defined in Eq. 8 as the criticality score. We then select the top 10% cells on each path as timing-critical cells and plug them into the same Enhanced Efficient-TDP pipeline.

Composite heuristic. The second baseline combines local delay, local net fanout, and geometric detour. For a cell c on path p, we define

$$g_1(c, p) = d_{\text{local}}(c, p), \tag{9}$$

$$g_2(c, p) = \log\left(1 + \max\left\{\text{fanout}(n_{\text{prev}}(c, p)), \text{fanout}(n_{\text{next}}(c, p))\right\}\right), \tag{10}$$

$$g_3(c, p) = \left\|\mathbf{r}_{\text{out}}(c_{\text{prev}}) - \mathbf{r}_{\text{in}}(c)\right\|_1 + \left\|\mathbf{r}_{\text{out}}(c) - \mathbf{r}_{\text{in}}(c_{\text{next}})\right\|_1, \tag{11}$$

where $n_{\text{prev}}(c, p)$ and $n_{\text{next}}(c, p)$ denote the predecessor and successor nets of cell c along path p, respectively. The local net fanout term therefore captures the larger fanout between the incoming and outgoing nets adjacent to the cell on the timing path. The geometric-detour term measures the local incoming and outgoing interconnect distance around the cell, where $\mathbf{r}_{\text{in}}(\cdot)$ and $\mathbf{r}_{\text{out}}(\cdot)$ denote

Table 8: **Placement runtime under the Efficient-TDP-based placement setting.** Runtime is reported in seconds.

Design	Efficient-TDP	Ours (TDP)
superblue1	1112.175	1334.025
superblue3	1263.597	1509.622
superblue4	1791.035	1629.099
superblue5	1365.149	1454.360
superblue7	1557.542	1455.127
superblue10	2770.350	2515.805
superblue16	722.887	1313.998
superblue18	512.626	514.316

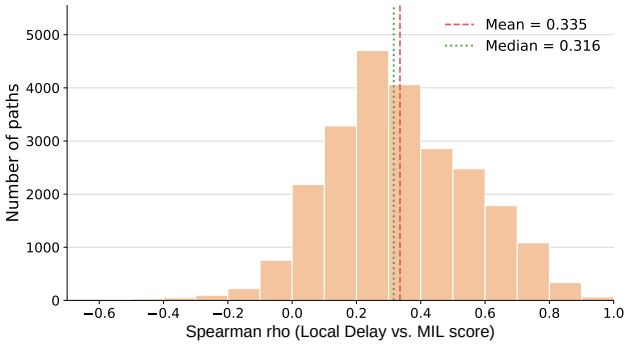


Figure 4: Distribution of per-path Spearman correlations between local-delay scores and MIL-predicted cell criticality scores on timing-critical paths from `ICCAD2015`. The red dashed line and green dotted line indicate the mean and median, respectively.

the physical locations of the input and output pins, respectively. For boundary cells on a path, the missing neighboring net or distance term is omitted.

For each path, we apply min–max normalization to each score:

$$\widetilde{g}_k(c,p) = \frac{g_k(c,p) - \min_{c' \in p} g_k(c',p)}{\max_{c' \in p} g_k(c',p) - \min_{c' \in p} g_k(c',p) + \epsilon}, \tag{12}$$

where ϵ is a small constant for numerical stability. The composite score is then computed as

$$s_{\text{comp}}(c,p) = \lambda_1 \widetilde{g}_1(c,p) + \lambda_2 \widetilde{g}_2(c,p) + \lambda_3 \widetilde{g}_3(c,p). \tag{13}$$

The configuration used in the main heuristic comparison is $(\lambda_1, \lambda_2, \lambda_3) = (0.8500, 0.0750, 0.0750)$. This setting gives the largest weight to local delay and equal weights to local net fanout and geometric detour. As with the local-delay heuristic, we select the top 10% cells on each path and use them in the same downstream Enhanced Efficient-TDP pipeline.

Downstream timing results. Table 9 reports the downstream WNS and TNS results when the heuristic critical cells are plugged into the same Enhanced Efficient-TDP pipeline. Both heuristic baselines provide useful timing signals and improve average TNS over the original Efficient-TDP baseline. However, they degrade WNS on average. In contrast, our MIL-guided method improves both WNS and TNS, achieving 4.08% WNS improvement and 10.56% TNS improvement over Efficient-TDP, as reported in Table 2. These results suggest that simple local heuristics capture only part of the cell-level timing signal, whereas learning from path-level supervision provides more effective fine-grained guidance for timing optimization.

Sensitivity analysis of composite heuristic weights. We further evaluate whether the composite heuristic baseline is sensitive to the manually specified weights. The main-comparison configuration $(\lambda_1, \lambda_2, \lambda_3) = (0.8500, 0.0750, 0.0750)$ assigns the largest weight to local delay, with equal auxiliary

Table 9: Comparison with non-ML cell-criticality heuristics under the Efficient-TDP-based setting. Composite uses $(\lambda_1, \lambda_2, \lambda_3) = (0.8500, 0.0750, 0.0750)$.

	Local-delay heuristic		Composite heuristic	
Design	WNS	TNS	WNS	TNS
superblue1	-8.20	-16.53	-7.89	-16.90
superblue3	-11.65	-19.23	-11.96	-19.19
superblue4	-9.01	-87.08	-8.91	-88.43
superblue5	-25.29	-62.03	-25.00	-63.77
superblue7	-15.40	-34.75	-15.40	-34.55
superblue10	-23.58	-561.42	-24.01	-567.20
superblue16	-8.85	-17.37	-8.06	-15.54
superblue18	-8.16	-15.84	-7.42	-16.56
Improvement	-3.63%	6.48%	-0.92%	6.05%

WNS and TNS use the same units as Table 2. The Improvement row reports average relative gains over the original Efficient-TDP baseline.

Table 10: Sensitivity analysis of composite heuristic weights.

$(\lambda_1, \lambda_2, \lambda_3)$	Avg. WNS Imp. (%)	Avg. TNS Imp. (%)
$(0.2000, 0.4000, 0.4000)$	-1.11	3.46
$(0.3000, 0.3500, 0.3500)$	-2.49	-0.98
$(0.3333, 0.3333, 0.3333)$	0.49	-7.94
$(0.5000, 0.2500, 0.2500)$	-0.74	4.88
$(0.6000, 0.2000, 0.2000)$	-1.15	5.16
$(0.6667, 0.1667, 0.1667)$	-2.01	4.99
$(0.7500, 0.1250, 0.1250)$	-1.61	6.02
$(0.8500, 0.0750, 0.0750)$	-0.92	6.05
$(0.9000, 0.0500, 0.0500)$	-2.25	5.95
$(0.6667, 0.0833, 0.2500)$	-0.28	5.05
$(0.6667, 0.1800, 0.1533)$	-1.22	5.47
$(0.6667, 0.2500, 0.0833)$	-0.94	5.80
$(0.8500, 0.0500, 0.1000)$	-2.03	6.09
$(0.8500, 0.1000, 0.0500)$	-4.94	3.98

All results report average relative improvements over the original Efficient-TDP baseline on `ICCAD2015`. Higher values are better.

weights for local net fanout and geometric detour. To test whether the comparison depends on this particular choice, we evaluate a range of alternative weight settings under the same Efficient-TDP-based placement pipeline.

Table 10 reports the average WNS and TNS improvements over the original Efficient-TDP baseline. The results show that changing the composite weights does not close the performance gap between the heuristic baseline and our MIL-guided method. Most heuristic variants improve TNS to some extent, but they generally degrade WNS. The largest TNS improvement among the configurations reported in Table 10 is 6.09%, achieved with $(\lambda_1, \lambda_2, \lambda_3) = (0.8500, 0.0500, 0.1000)$, compared with 10.56% for our MIL-guided method. Moreover, none of the heuristic variants simultaneously improves both WNS and TNS as effectively as our method.

C.5 Path-Level Validation Using Identified Timing-Critical Cells

Beyond comparisons with handcrafted heuristics, we further perform a path-level consistency check using the identified timing-critical cells. Although our model is trained with path-level supervision and is primarily used for cell-level attribution, a useful validation is whether the inferred critical cells can recover whether a path is timing-critical.

Table 11: Path-level validation using the identified timing-critical cells on ICCAD2015.

Confusion statistic	TP	FP	TN	FN
Count	7493	2345	5655	507
Metric	Accuracy	Precision	Recall	F1
Value	0.8218	0.7616	0.9366	0.8401

For each circuit in ICCAD2015, we extract 1000 timing-critical paths and 1000 non-critical paths. For each path, we apply our MIL model to identify timing-critical cells using the same cell-selection rule as in the downstream placement experiments. We then classify a path as timing-critical if at least one cell on the path is identified as timing-critical; otherwise, the path is classified as non-critical. Table 11 reports the aggregated results across all eight circuits.

These results assess the consistency of the model's path-level predictions with STA-derived labels.

C.6 Case Study of Identified Timing-Critical Cells

As additional qualitative evidence for the learned cell-level attribution, we visualize several representative timing-critical paths together with the cells identified as timing-critical by our MIL model. Because standard STA flows do not provide direct cell-level criticality labels, evaluation of timing-critical cell identification necessarily relies on indirect evidence.

Figure 5 presents four representative timing-critical paths extracted from the DREAMPlace placement results of superblue1, superblue3, superblue4, and superblue5, respectively. In each subfigure, the gray polyline denotes the timing path, green circles denote non-critical cells, orange circles denote the cells identified as timing-critical by our model, and diamond markers denote endpoint pins.

These examples provide intuitive support for the learned cell-level attribution. We observe that the identified timing-critical cells often appear at locations with pronounced geometric detour or strong local structural irregularity, suggesting that detour is indeed an important factor for cell criticality. At the same time, not all cells with visible detour are identified as timing-critical, indicating that detour alone is insufficient to fully characterize cell-level timing bottlenecks. This observation is consistent with our earlier quantitative analysis: the learned criticality scores are related to local delay and geometric effects, but are not reducible to a single handcrafted factor. Overall, these case studies suggest that the proposed MIL framework is able to identify physically meaningful bottleneck cells from path-level supervision.

C.7 Robustness to an Alternative Data Split

To further examine the robustness of the proposed timing-critical cell identification framework, we conduct an additional experiment using an alternative random circuit split. Specifically, we use superblue3, superblue5, and superblue10 for training, superblue7 and superblue18 for validation, and superblue1, superblue4, and superblue16 for zero-shot testing. After training the MIL model under this split, we plug the identified timing-critical cells into the same Enhanced Efficient-TDP pipeline.

Table 12 reports the resulting WNS and TNS on the ICCAD2015 benchmarks. The results remain close to those obtained under the original split, indicating that the effectiveness of the proposed cell-level timing attribution is not tied to a particular circuit split. Compared with the original Efficient-TDP baseline in Table 2, this alternative split achieves average improvements of 4.26% in WNS and 8.22% in TNS.

D Visualization of Placement

We visualize the outcomes of Timing-Critical-Cell-Guided Clustering-Based Placement and enhanced Efficient-TDP on eight ICCAD2015 circuits in Figures 6 and 7, respectively.

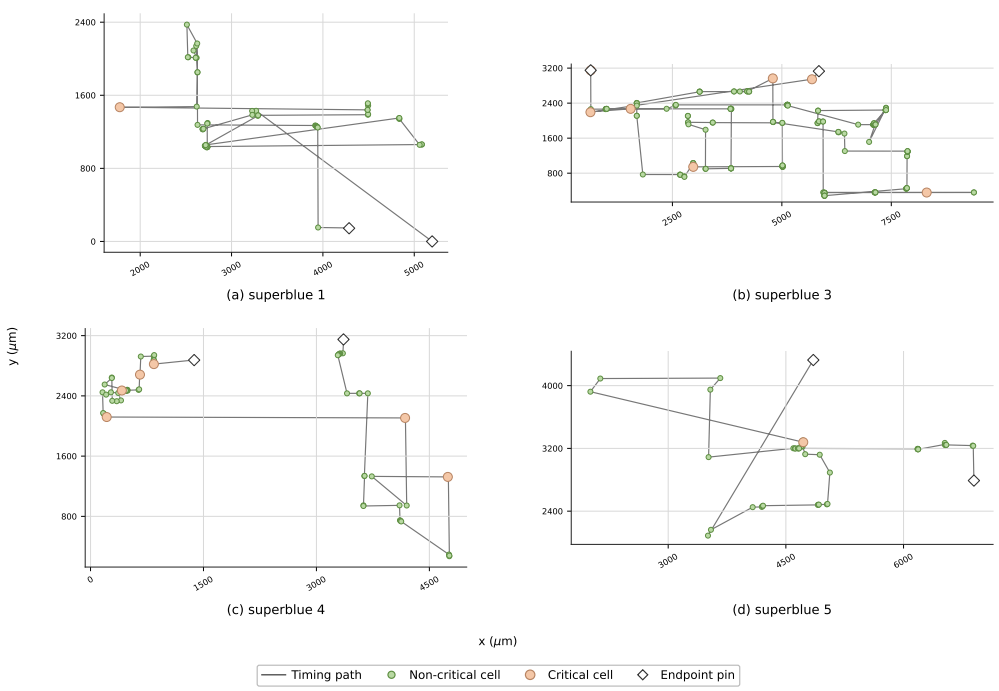


Figure 5: Case studies of timing-critical cells identified by the proposed MIL model on representative timing-critical paths from the DREAMPlace placement results of `superblue1`, `superblue3`, `superblue4`, and `superblue5`. The gray polyline denotes the timing path, green circles denote non-critical cells, orange circles denote the identified timing-critical cells, and diamond markers denote endpoint pins. The visualizations show that the learned timing-critical cells often coincide with path segments exhibiting noticeable detour or locally unfavorable geometric configurations, while not being fully explained by detour alone.

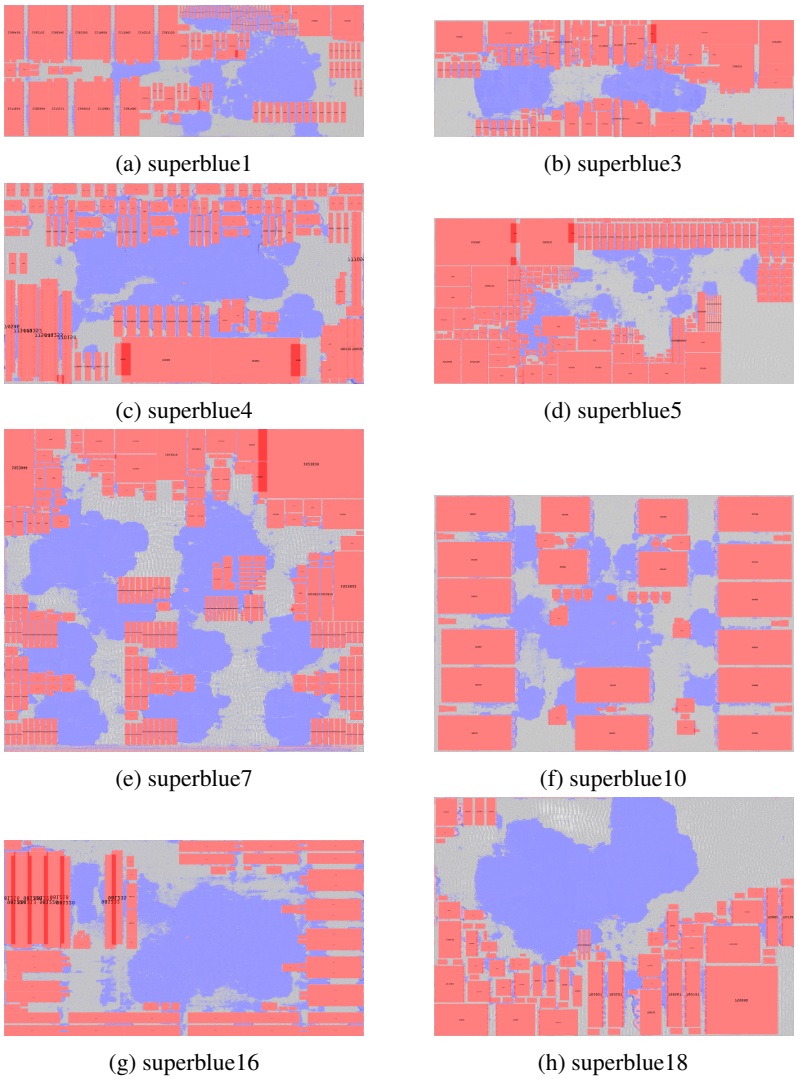
Table 12: Enhanced Efficient-TDP results using an alternative random data split. WNS and TNS use the same units as Table 2.

Design	WNS	TNS
superblue1	-7.28	-16.15
superblue3	-11.21	-19.23
superblue4	-8.09	-85.55
superblue5	-24.98	-61.31
superblue7	-15.40	-31.66
superblue10	-22.50	-556.05
superblue16	-8.51	-18.48
superblue18	-6.25	-14.94
Improvement	4.26%	8.22%

E Discussions

E.1 Broader Impact

The broader significance of this work lies in learning fine-grained optimization guidance from supervision that describes only aggregated system outcomes. Such feedback does not directly reveal which components should be prioritized for improvement. Through timing optimization in chip design, our results show that coarse supervision can nevertheless yield component-level guidance useful for downstream optimization, without requiring component-level annotations. This perspective



(a) superblue1 (b) superblue3

(c) superblue4 (d) superblue5

(e) superblue7 (f) superblue10

(g) superblue16 (h) superblue18

Figure 6: Visualization of Timing-Critical-Cell-Guided Clustering-Based Placement on eight ICCAD2015 circuits.

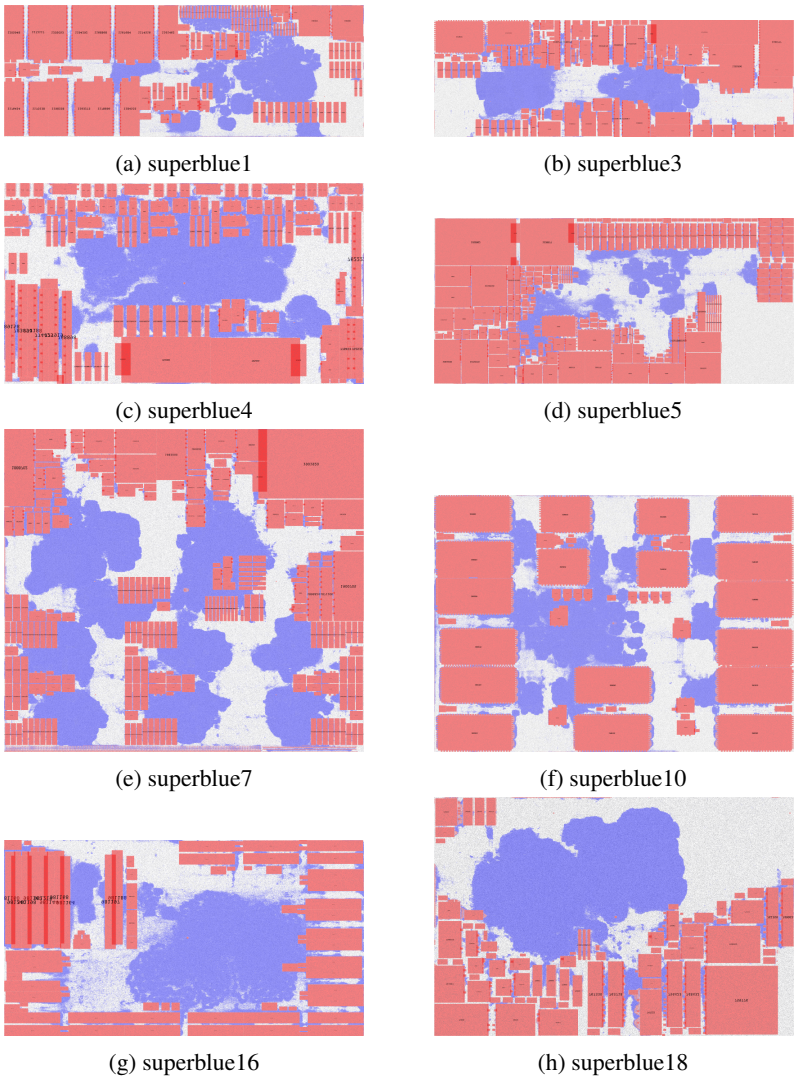
may inform the design of learning frameworks for other settings in which the granularity of available feedback differs from that of optimization decisions.

Within chip design, CritiCell makes existing path-level timing feedback more actionable by learning cell-level importance scores. Its integration into both constraint-driven and objective-driven placement illustrates how learned guidance can support different optimization mechanisms. Beyond placement, similar formulations may be explored for gate sizing, buffering, and routing, extending the study of fine-grained guidance to other stages of timing optimization.

E.2 Limitations and Future Directions

While CritiCell demonstrates promising empirical results, several directions remain open for future exploration.

Beyond global placement. This work focuses on global placement as the main application scenario. A natural future direction is to extend the same attribution principle to other stages of the physical design flow, such as buffering, gate sizing, detailed placement, and routing. In these stages, fine-



(a) superblue1 (b) superblue3

(c) superblue4 (d) superblue5

(e) superblue7 (f) superblue10

(g) superblue16 (h) superblue18

Figure 7: Visualization of enhanced Efficient-TDP guided by timing-critical cells on eight ICCAD2015 circuits.

grained identification of dominant contributors to timing degradation may further enable targeted local optimization and improve timing closure across the full design flow.

Toward multi-objective fine-grained attribution. The current instantiations of CritiCell primarily focus on timing optimization, with WNS and TNS as the main evaluation metrics. In practical chip design, however, physical design also needs to consider other important objectives, including power and area. Future work can extend the proposed framework from timing-critical cell identification to multi-objective cell-level attribution, where the model identifies cells that are critical to different design objectives. Such fine-grained identification would help downstream optimization distinguish cells that dominate different aspects of design-quality degradation, making cell-level attribution more broadly useful for practical physical design optimization and supporting more balanced PPA-aware decision making.