Quality-Preserving Legalization for Analytical Floorplanning

Yunqi He
yunqi.he@u.northwestern.edu
Northwestern University
Evanston, IL, USA

You Li✉
you.li@u.northwestern.edu
Northwestern University
Evanston, IL, USA

Ruofan Huang
ruofanhuang@u.northwestern.edu
Northwestern University
Evanston, IL, USA

Hai Zhou
haizhou@northwestern.edu
Northwestern University
Evanston, IL, USA

Abstract

Floorplanning is a crucial step in VLSI physical design. Analytical floorplanning, which determines macro positions through electrostatic-based mixed-size placement, has become the most prevalent paradigm. However, the resulting floorplan contains inter-macro overlaps that must be resolved by a legalization step. Existing methods minimize displacement from the analytical floorplan without effective topological guidance, leading to inconsistent downstream quality. In this paper, we propose a new legalization workflow that preserves the high quality of analytical floorplanning. We leverage twin binary trees to capture the relation between adjacent blocks, and build a differentiable model to optimize their coordinates. Experiments on TILOS MacroPlacement benchmarks show that our method enables analytical floorplanning to match or surpass commercial and open-source macro placement tools in post-route timing and wirelength.

ACM Reference Format:
Yunqi He, You Li, Ruofan Huang, and Hai Zhou. 2026. Quality-Preserving Legalization for Analytical Floorplanning. In *IEEE/ACM International Conference on Computer-Aided Design (ICCAD '26), November 08–12, 2026, San Jose, CA, USA.* ACM, New York, NY, USA, 7 pages. https://doi.org/10.1145/3831252.3833939

1 Introduction

Macro placement plays a critical role in VLSI physical design. Hard IP blocks, including processor cores, memory blocks, and I/O interfaces, typically constitute 30% to 60% of the core area in modern SoCs. Their positions in the layout significantly impact the quality of results (QoR), which is measured by wirelength, timing closure, power distribution, and routing congestion. The state-of-the-art methods are electrostatic-based analytical placers. Pioneering works like ePlace [15] and RePlAce [5] propose to co-optimize macros and standard cells within the same global density field. Recent advances like DREAMPlace [12, 13], Xplace [14], and AutoDMP [1] further improve speed and quality with GPU acceleration. An analytical floorplan encodes two types of information, which collectively determine the final QoR: (1) the *positions* of macros, which are optimized to minimize wirelength under density constraints, and (2) the *topology* of the layout, which describes the spatial relations of macros and standard cell clusters. Any disruptions to either the positions or the topology will degrade the final PPA results.

Legalization as a bottleneck. The aforementioned placers can produce overlapping macros. Legalization is necessary for design to proceed, and it has become a major bottleneck in physical design [1]. Existing legalizers minimize displacement from the analytical floorplan without effective topological guidance. Greedy methods [8, 13] sequentially place every macro in the closest legal position. As a result, the topology after legalization may be significantly changed from the placement solution. Similarly, simulated annealing (SA) methods [2] can fundamentally change the original solution, thereby deteriorating the QoR. Linear programming (LP) methods [4, 7, 19] assign a non-overlapping constraint to each pair of macros and minimize the total displacement under these constraints. However, such constraint graphs are highly redundant: an unnecessary constraint can still be added for two distant macros [22]. This redundancy over-constrains the LP problem and increases its complexity. In contrast, the essential adjacency information is absent from the constraint graph, which complicates the estimation of interconnection and area.

Reinforcement learning (RL) based macro placement methods tackle legalization from a different perspective. AlphaChip [6, 16] trains an RL agent to place macros on a discretized grid to avoid overlaps. Subsequent efforts [11, 20] have improved placement quality by leveraging more effective representation and optimization techniques. Although grid-based methods can prevent overlaps by construction, their restrictions on the solution space have a negative impact on the quality of placement. This impact can be mitigated with an additional refinement step that is similar to legalization.

Fig. 1 analyzes different legalization methods on ariane133, a RISC-V design with 132 macros. It investigates 5 data cache macros to illustrate how each method behaves. Compared with the original floorplan, the greedy method flips the topological order of 3 pairs of macros. The LP method retains all topological orders, but it almost doubles the total displacement. Our method retains all topological orders and achieves the lowest total displacement. Meanwhile, it achieves a worst negative slack (WNS) of -0.20 ns, which is optimal among the 3 solutions. The rationale behind our method is to preserve both the positions and the topology of the original placement solution as much as possible.

Our method. Sequence pair [17], B*-tree [3], and twin binary trees [21] are commonly used data structures to represent circuit floorplans. Sequence pair does not explicitly encode adjacency information. B*-tree encodes partial adjacency through its tree structure, but as a single binary tree it cannot capture the complete adjacency relations among all blocks. Both data structures rely on randomized techniques such as SA to search for a new floorplan. In this process, the topology can radically change from the previous version. The twin binary trees (TBT) representation is a non-redundant data structure that can encode and decode a mosaic floorplan in $O(n)$ time. In contrast to the sequence pair and the B*-tree, TBT uses two complementary binary trees to completely capture the adjacency information of a mosaic floorplan. With the constraint graphs of TBT, the coordinates of the blocks are continuous functions of their sizes. This property allows gradient-based, controllable optimization of floorplans.

Our method aims to preserve the high-quality placement results during legalization in terms of both the positions and the topology. First, we apply *mosaic construction* to obtain a rectangular partitioning of the original layout. Each rectangle is either a *macro block* or a *filler block*. Afterward, we use the MFTB (mosaic floorplan to twin binary trees) algorithm to encode the mosaic floorplan into a TBT (τ^+, τ^-). Finally, we use the TBMF (twin binary trees to mosaic floorplan) algorithm to reconstruct the layout from the constraint graphs of the TBT. Mosaic construction and MFTB are executed once during preprocessing, and the final stage is repeated during optimization. Notably, the coordinates in each reconstructed layout are *piecewise-linear* functions of the dimensions of filler blocks. As such, we build the first *differentiable* TBT-based model for layout optimization. Compared

ICCAD '26, San Jose, CA, USA

ACM ISBN 979-8-4007-2873-0/2026/11
https://doi.org/10.1145/3831252.3833939

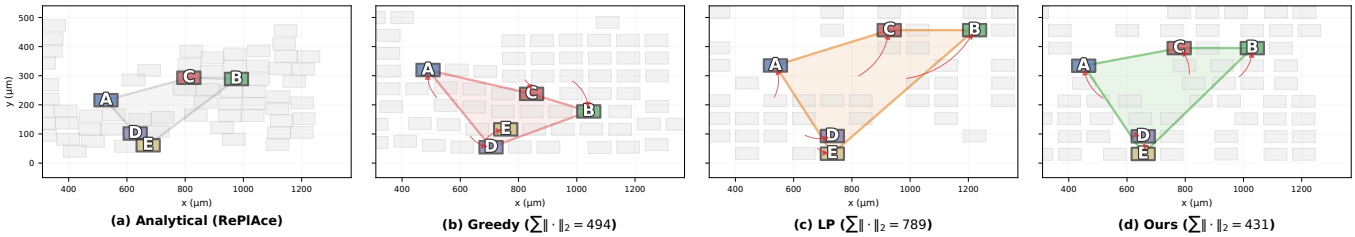

Figure 1: Analyzing the positions and topology of 5 data cache macros in ariane133 RISC-V core. (a) Placement produced by analytical placer RePlAce. The convex hull is shown in gray. (b) Greedy method flips 3 pairs of topological orders ($\sum\|\cdot\|_2$=494). (c) LP method preserves all orders but incurs a large displacement ($\sum\|\cdot\|_2$=789). (d) Our method preserves all orders with low displacement ($\sum\|\cdot\|_2$=431). Red arrows represent the displacements before and after legalization.

with LP methods, our method can guarantee the legality and feasibility of each reconstructed layout throughout optimization. Compared with SA methods, it converges in merely hundreds of iterations and requires minimal hyperparameter tuning.

Our main contributions are summarized as follows:

- We identify *topology preservation* as a key factor of legalization quality. Experiments on commercial and open-source design flows have confirmed this observation.
- We construct a *piecewise linear* model to optimize the coordinates in a physical layout. We ensure that the layout remains legal throughout the optimization process.
- We develop a comprehensive legalization workflow that preserves the quality of electrostatic placers. It also demonstrates the usefulness of TBT in VLSI physical design.

2 Preliminaries

This section reviews the technical foundations of our method: electrostatic analytical placement, which produces the initial placement that legalization aims to preserve, and the twin binary tree representation, which provides the structural backbone of our legalization procedure.

2.1 Electrostatics-Based Analytical Placement

Modern analytical mixed-size placers formulate placement as an unconstrained optimization problem [5, 15]:

$$\min_{\mathbf{v}}; W(\mathbf{v}) + \lambda, D(\mathbf{v}), \tag{1}$$

where $\mathbf{v}$ collects the coordinates of all movable objects (macros and standard cells), W is the HPWL wirelength objective, D is a density penalty, and λ balances the two terms.

The density penalty adopts an electrostatic analogy: each movable object i is modeled as a positive charge q_i proportional to its area. The resulting charge density $\rho(x, y)$ and electric potential $\psi(x, y)$ satisfy Poisson's equation with Neumann boundary conditions:

$$\nabla^2\psi(x, y) = -\rho(x, y), \qquad \hat{n} \cdot \nabla\psi\big|_{\partial R} = 0. \tag{2}$$

The density penalty is the total electrostatic potential energy, $D = \frac{1}{2}\sum_i q_i\psi_i$. Its gradient with respect to object i's position is $q_i\nabla\psi_i$, so the corresponding electrostatic force is $-q_i\nabla\psi_i$, which repels overlapping objects. Solving Poisson's equation via FFT on a placement grid yields a smooth global density field that captures interactions among all movable objects.

ePlace-MS [15] optimizes Eq. (1) using Nesterov's method, balancing wirelength and density for macros and standard cells. RePlAce [5] further incorporates local bin overflow into its density function, enabling constraint-oriented local smoothing with low runtime overhead.

2.2 Twin Binary Trees

A *mosaic floorplan* [21] is a partition of the chip area into non-overlapping rectangles (called *blocks*). Every internal boundary segment terminates in a **T-junction**: no two segments form a cross.

Yao et al. [21] established a one-to-one correspondence between mosaic floorplans with n blocks and pairs of binary trees called *twin binary trees*:

Definition 1 (Twin Binary Trees [21]). *Let* Tree_n *denote the set of binary trees with n nodes. The* augmented tree t^* *is obtained by adding a leaf labeled* 0 *for each missing left child and a leaf labeled* 1 *for each missing right child. The* 2-labeling $\mathcal{L}(t)$ *is the sequence of these augmented leaf labels under in-order traversal of* t^* *(omitting the first and last bits). The set of twin binary trees* TBT_n *consists of all pairs* $(t_1, t_2) \in \mathrm{Tree}_n \times \mathrm{Tree}_n$ *satisfying:*
(1) $\mathcal{I}(t_1) = \mathcal{I}(t_2)$*: both trees share the same in-order traversal;*
(2) $\mathcal{L}(t_1) = \mathcal{L}^c(t_2)$*: the 2-labelings are bitwise complements.*

The complement condition implies that children are distributed complementarily between the two trees: if a node has a left child in t_1, it has no left child in t_2 (likewise for right children).

Two algorithms make this correspondence constructive. Following Yao et al., we denote the two trees as τ^+ and τ^-:

MFTB (Mosaic Floorplan → Twin Binary Trees). Given a mosaic floorplan, each block's upper-right corner T-junction determines its parent in τ^+, and its lower-left corner T-junction determines its parent in τ^-. From the parent relationships, the two trees can be constructed in $O(n)$ time.

TBMF (Twin Binary Trees → Mosaic Floorplan). Given (τ^+, τ^-) and block sizes (w_i, h_i), the TBMF algorithm constructs horizontal and vertical constraint graphs G_H, G_V from the augmented twin binary trees. Each internal boundary of the mosaic becomes a *vertex*, and each block becomes a directed *edge* connecting its two opposing boundaries, weighted by the block's dimension. Boundary coordinates are computed via *longest-path* on these DAGs. Specifically, for a vertex v with incoming edges $e = (v' \to v)$ of weight w_e,

$$x_v = \max_{e=(v'\to v)\,\in\, G_H} (x_{v'} + w_e), \tag{3}$$

$$y_v = \max_{e=(v'\to v)\,\in\, G_V} (y_{v'} + w_e). \tag{4}$$

The constraint graphs are constructed in $O(n)$ time and guarantee that no two blocks overlap.

3 Methodology

3.1 Overview

Given overlapping macro positions from an analytical placer, our framework produces a legal floorplan that preserves both the positions and topology of the original solution (Fig. 2). We require the analytical placer to run on *padded* macro dimensions that are aware of minimum spacing. We first apply *uniform shrinking* to the padded dimensions, producing a compact layout suitable for mosaic construction. This layout is partitioned into a mosaic floorplan of macro blocks and filler blocks. The *MFTB* algorithm then encodes the mosaic floorplan into twin binary trees, fixing the topology. Finally, with the topology fixed, we iteratively optimize macro positions to minimize ℓ^2 displacement.

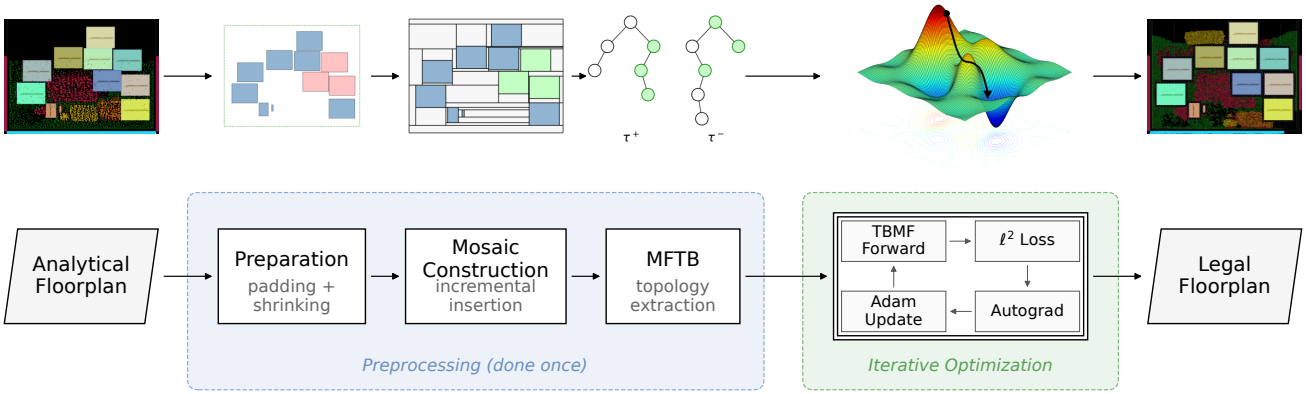

Figure 2: End-to-end pipeline of our legalization framework, illustrated on bp_fe_top (nangate45). Preprocessing (done once) constructs the mosaic floorplan and extracts twin binary trees; iterative optimization minimizes ℓ^2 displacement via differentiable TBMF.

The forward model is derived from the *TBMF* algorithm. The first few stages run once as preprocessing; only the last stage iterates.

3.2 Preparation

3.2.1 Macro Padding. We pad macro dimensions before running the analytical placer so that minimum spacing is incorporated into the resulting floorplan. Following AutoDMP [1], we add a uniform halo to each macro in the LEF description. The halo enforces minimum spacing between macros, allowing the analytical floorplan to capture spacing-aware spatial relations. For designs whose macros differ greatly in size, the analytical placer's density field may poorly resolve overlaps among the smallest macros. We address this by applying per-type inflation to these macros, keeping the maximum pairwise overlap ratio below a controlled threshold. This conditioning helps ensure a reasonable shrink ratio during subsequent uniform shrinking. The resulting macro blocks are comparable in size to the filler blocks, preventing fillers from dominating the constraint graph and keeping constraint chains manageable.

3.2.2 Uniform Shrinking. To prepare an input suitable for mosaic construction, we uniformly shrink all macro dimensions while keeping each macro's center fixed:

$$w_i' = s \cdot w_i, \quad h_i' = s \cdot h_i, \quad \mathbf{c}_i' = \mathbf{c}_i, \tag{5}$$

where w_i, h_i are the padded dimensions and the optimal scale factor $s^* \in (0, 1]$ is the largest s such that no pair of scaled macros overlaps. We find s^* via binary search in $O(n^2 \log(1/\epsilon))$ time. Macro orientations can be determined by the analytical placer (e.g., by modeling each macro as more than one charge), so orientation is not a legalization concern. Thus, we preserve the orientations from the analytical placer.

3.3 Mosaic Construction

Taking the compact layout from uniform shrinking as input, we construct a mosaic floorplan. We provide two variants: an incremental mosaic builder and a simplified slicing tree partitioner.

3.3.1 Incremental Mosaic Builder. Inspired by the corner-stitch data structure [18], the mosaic builder (Algorithm 1) starts with the entire chip area as a single filler block and inserts macros one by one. Each split extends only one pair of opposite edges, so the cuts do not form a full cross, creating T-junctions at the macro corners and ensuring the property required by MFTB. The pair to extend is chosen dynamically based on the cumulative partition span in each dimension, yielding a more balanced structure for subsequent optimization. Once all macros are inserted, adjacent fillers sharing a full edge are merged to reduce the total block count.

Algorithm 1 Incremental Mosaic Builder

Input: Shrunk macros $\{m_1, \ldots, m_n\}$, chip area R
Output: Mosaic blocks $\{b_1, \ldots, b_N\}$ (macros + fillers)

$\mathcal{R} \leftarrow \{R\}$ ▷ *Initialize with single filler*
for each macro m_i **do**
  $\mathcal{R}' \leftarrow \emptyset$
  for each rectangle $r \in \mathcal{R}$ **do**
    if r does not overlap m_i **then**
      $\mathcal{R}' \leftarrow \mathcal{R}' \cup \{r\}$
    else
      $e \leftarrow$ one pair of opposite edges of m_i
      Extend e to the boundary of r
      Cut r along e and the edges of m_i
      $\mathcal{R}' \leftarrow \mathcal{R}' \cup \{\text{resulting pieces}\}$
  $\mathcal{R} \leftarrow \mathcal{R}'$
Merge adjacent fillers sharing full edges
return $\mathcal{R}$

3.3.2 Slicing Tree as Simplified Alternative. For designs with simpler macro configurations, a *slicing tree partitioner* can replace the incremental mosaic builder. The slicing tree generates a *slicing floorplan*, a proper subset of mosaic floorplans. In our experiments, ariane133/136 in the OpenROAD flow uses the slicing tree partitioner, while all other benchmarks use the incremental mosaic builder.

3.4 MFTB: Mosaic to Twin Binary Trees

With the mosaic floorplan constructed, we encode its topology into twin binary trees (τ^+, τ^-) using the MFTB algorithm. Yao et al. describe a bottom-up construction that locates each block's parent via

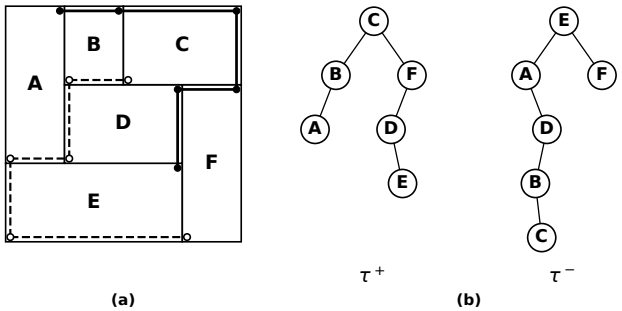

Figure 3: MFTB example: (a) a mosaic floorplan with six blocks; (b) the corresponding twin binary trees (τ^+, τ^-).

its corner T-junctions [21]; we instead implement an equivalent top-down recursion that descends from the root, identifying each block's children.

τ^+ *construction.* The root is the block at the upper-right (UR) corner of the mosaic floorplan. For each block b, the *left child* is the block to the left of b whose top edge aligns with b's top edge; the *right child* is the block below b whose right edge aligns with b's right edge.

τ^- *construction.* The root is the block at the lower-left (LL) corner. For each block b, the *left child* is the block above b whose left edge aligns with b's left edge; the *right child* is the block to the right of b whose bottom edge aligns with b's bottom edge.

The recursion terminates when no matching neighbor exists, producing a leaf node (Fig. 3). The resulting trees are then held fixed throughout the subsequent optimization.

3.5 TBMF as a Differentiable Forward Model

The previous stages capture the topology of the analytical floorplan as twin binary trees. We now feed these trees, together with the block sizes, into the TBMF algorithm to produce a legal floorplan. Since macro sizes are fixed by the design, filler sizes are the only remaining degrees of freedom. Adjusting them moves macros toward their analytical positions while leaving the topology intact, so both properties of the analytical floorplan carry over to the legal one. We first describe the reconstruction and prove that it guarantees overlap freedom and topological invariance, then show how to optimize filler sizes via gradient descent to minimize macro displacement.

3.5.1 From Twin Binary Trees to Mosaic Floorplan. As described in Section 2.2, TBMF derives constraint graphs G_H and G_V and computes block coordinates via longest-path. We now detail the construction.

The twin binary trees are first augmented by adding a virtual leaf for each missing child (Definition 1; Fig. 4). Each augmented leaf represents a boundary of its parent block: in τ^+, a left leaf represents the top boundary and a right leaf represents the right boundary; in τ^-, a left leaf represents the left boundary and a right leaf represents the bottom boundary. These boundaries become vertices in the constraint graphs G_H and G_V. Each block is sandwiched between two boundaries, forming a directed edge weighted by the block's width (in G_H) or height (in G_V). Summing edge weights along a path yields boundary coordinates (Eqs. 3–4), which in turn determine block coordinates.

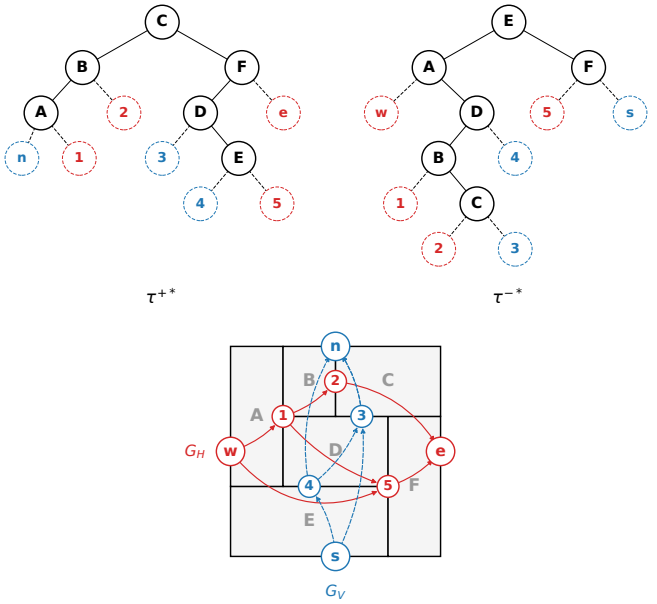

Figure 4: TBMF construction for the six-block example of Fig. 3.

3.5.2 Structural Guarantees. We now show the overlap freedom and topological invariance of the reconstruction.

Theorem 1 (Non-Overlapping Guarantee). *For a pair of twin binary trees (τ^+, τ^-) and any block size vector $\mathbf{s} \geq 0$, the TBMF-reconstructed floorplan contains no overlapping blocks.*

Proof. TBMF provides a one-to-one mapping from twin binary trees to mosaic floorplans (Theorem 2.1.15 of [21]). We refer readers to [21] for the full proof. A mosaic floorplan is by definition a partition into non-overlapping rectangles. □

Theorem 2 (Topological Invariance). *For a pair of twin binary trees (τ^+, τ^-) and any block size vector $\mathbf{s} \geq 0$, the relative positions of blocks in the TBMF-reconstructed floorplan are invariant. That is, the relative spatial relations among blocks do not depend on $\mathbf{s}$.*

Proof. The *directed ancestor set* of an augmented leaf k in tree t, denoted $D(k, t)$, consists of all nodes from which k can be reached by consecutive same-direction branches. By the properties of directed ancestor sets [21], for each augmented leaf k, $D(k, \tau^{+*})$ and $D(k, \tau^{-*})$ identify the blocks on either side of the boundary that k represents. Since $D(k, \tau^{+*})$ and $D(k, \tau^{-*})$ are determined entirely by the tree structure of (τ^+, τ^-), they are independent of $\mathbf{s}$. The blocks on each side of every boundary are therefore fixed for all $\mathbf{s} \geq 0$, and the relative spatial relations among all blocks are invariant. Completeness follows from Theorem 2.1.15 of [21]. □

Theorem 2 is the formal basis of our quality preservation claim. The filler blocks represent inter-macro regions—typically occupied by standard-cell clusters in the analytical floorplan. Optimizing filler sizes changes *where* macros sit but not their relative positions.

3.5.3 Displacement Optimization. Under these guarantees, we restore macro sizes to their real dimensions (original dimensions plus the spacing halo) and optimize filler sizes to minimize displacement from the original analytical floorplan. A key observation is that once the block ordering is determined by the constraint graphs, the remaining coordinate computation consists only of additions and max operations. This computation is *piecewise linear* in filler sizes and therefore differentiable almost everywhere, so the longest-path computation of TBMF serves as a forward model for gradient-based optimization. During backpropagation, each max node passes the gradient to the predecessor on the current longest path, analogous to how a ReLU gate routes gradients.

We minimize the total ℓ^2 displacement of macro centers:

$$\mathcal{L}(\mathbf{f}) = \sum_{i=1}^{n} \left\| \mathbf{c}_i^{\text{tbmf}}(\mathbf{f}) - \mathbf{c}_i^{\text{init}} \right\|_2, \tag{6}$$

where $\mathbf{f} = \{(w_f, h_f)\}_{f \in F}$ are the filler block sizes (the only optimization variables), $\mathbf{c}_i^{\text{tbmf}}$ is the macro center computed by TBMF, and $\mathbf{c}_i^{\text{init}}$ is the original analytical center.

4 Experiments

To evaluate our quality-preserving legalization framework, we design experiments to answer the following questions:

RQ1 (Topology Preservation): Does preserving the topology during displacement minimization yield more consistent post-route PPA?

RQ2 (End-to-End Competitiveness): Does our legalization enable analytical floorplanning to compete with state-of-the-art macro placement approaches?

RQ3 (Design Choices): How does each stage of our pipeline contribute to the final quality, and does the optimization converge stably?

RQ4 (Efficiency): How do the legalization runtime and the end-to-end pipeline runtime scale with design complexity?

Table 1: Legalization comparison on 8 OpenROAD nangate45 designs. All three methods legalize the same RePlAce output. LP requires enlarged die on 6 of 8 designs (overhead shown in Die+ column). Disp = ℓ^2 displacement; WNS = worst negative slack; TNS = total negative slack; rWL = routed wirelength. "N/A": downstream flow failed. Best values in **bold**.

Design	#M	Die+ (LP)	Mean/Max Disp (μm)			WNS (ns)			TNS (ns)			rWL (Mμm)			Power (mW)		
			Ours	Greedy	LP	Ours	Greedy	LP	Ours	Greedy	LP	Ours	Greedy	LP	Ours	Greedy	LP
ariane133	132	+44%	85/288	**65/203**	217/374	**−0.20**	−0.32	−0.21	−142	−238	**−77**	5.83	5.77	**5.74**	384	383	**381**
ariane136	136	+42%	**101/246**	115/370	225/477	**0**	**0**	**0**	**0**	**0**	**0**	6.64	6.56	**6.46**	311	**307**	**307**
black_parrot	24	+12%	**22/56**	27/113	32/76	−0.45	N/A	**−0.19**	−0.5	N/A	**−0.2**	**7.78**	N/A	8.13	**200**	N/A	201
bp_be_top	10	+4%	22/41	54/154	**21/36**	**0**	**0**	**0**	**0**	**0**	**0**	2.08	**2.05**	2.09	75.7	**75.0**	75.6
bp_fe_top	11	0%	**11/27**	31/145	12/31	**0**	**0**	**0**	**0**	**0**	**0**	1.41	**1.40**	1.45	56.7	**56.6**	56.8
bp_multi_top	26	+29%	**42/87**	58/199	73/117	−2.06	**−1.85**	−2.01	−2.1	**−1.9**	−2.0	**3.50**	3.64	4.02	**203**	204	207
swerv_wrapper	28	+25%	14/**26**	9/145	**8/27**	**0**	**0**	−0.11	**0**	**0**	−2.5	3.88	**3.76**	4.79	411	**408**	426
tinyRocket	2	0%	**3/5**	**3/5**	9/17	**0**	**0**	**0**	**0**	**0**	**0**	0.60	**0.59**	**0.59**	**186**	**186**	187

Table 2: End-to-end macro placement comparison on 8 OpenROAD nangate45 designs. Best values in **bold**.

Benchmark	#M	WNS (ns)			TNS (ns)			rWL (Mμm)			Power (mW)		
		Ours	Hier-RTLMP	Triton	Ours	Hier-RTLMP	Triton	Ours	Hier-RTLMP	Triton	Ours	Hier-RTLMP	Triton
ariane133	132	**−0.20**	−0.49	−0.25	**−141.7**	−609.2	−184.4	**5.83**	6.82	6.40	**384**	389	386
black_parrot	24	**−0.45**	−0.81	−0.67	**−0.45**	−0.81	−0.67	**7.78**	8.22	8.07	**200**	206	**200**
bp_be_top	10	**0**	**0**	**0**	**0**	**0**	**0**	**2.08**	2.29	2.34	75.7	76.5	**75.3**
bp_fe_top	11	**0**	**0**	**0**	**0**	**0**	**0**	1.41	**1.34**	1.49	56.7	**55.1**	55.8
bp_multi_top	26	−2.06	**−1.81**	−1.94	−2.06	−16.78	**−1.94**	**3.50**	3.86	4.01	**203**	205	206
swerv_wrapper	28	**0**	−0.08	**0**	**0**	−1.90	**0**	3.88	3.99	**3.75**	411	**392**	396
tinyRocket	2	**0**	**0**	**0**	**0**	**0**	**0**	**0.60**	0.61	0.61	186	186	**185**
ariane136	136	**0**	**0**	**0**	**0**	**0**	**0**	**6.64**	7.04	6.75	311	**303**	305

Table 3: End-to-end macro placement comparison (Cadence Innovus). NanGate45 baselines from TILOS [6] (Innovus 21.10); Ours and CMP† run on Innovus 21.19. ariane133* has 133 macros, distinct from the 132-macro OpenROAD version. Best values in **bold**.

Method	ariane133* NG45 (1.3 ns)				bp_quad NG45 (1.3 ns)				ariane133* ASAP7 (0.9 ns)				bp_quad ASAP7 (0.85 ns)			
	WNS	TNS	rWL	Pwr	WNS	TNS	rWL	Pwr	WNS	TNS	rWL	Pwr	WNS	TNS	rWL	Pwr
Ours	−100	−75.8	4.41	831	−221	−1228	26.4	4488	−12	−0.8	0.93	505	−113	−792	7.00	1544
CMP†	−167	−143	3.97	811	−204	−964	23.8	4444	−12	−0.7	0.93	505	−116	−632	6.01	1526
CMP	−154	−197	4.06	852	−144	−356	**23.1**	**4429**	−124	−146	**0.84**	504	−111	**−240**	**6.10**	1529
SA	−126	−117	**3.98**	**827**	−230	−3013	28.9	4512	−124	−141	**0.89**	**503**	−120	−425	7.27	1547
CT_AC	**−88**	−52	5.00	836	−230	−1487	33.2	4569	−108	−105	1.01	505	−201	−1449	8.88	1569
CT_Scratch	−140	−119	4.65	832	−205	−1203	47.8	4822	−142	−184	1.03	505	−226	−2044	11.4	1609
Human	**−88**	**−47**	4.68	832	**−97**	**−322**	25.9	4470	**−104**	**−82**	1.18	508	**−89**	−357	6.52	1536

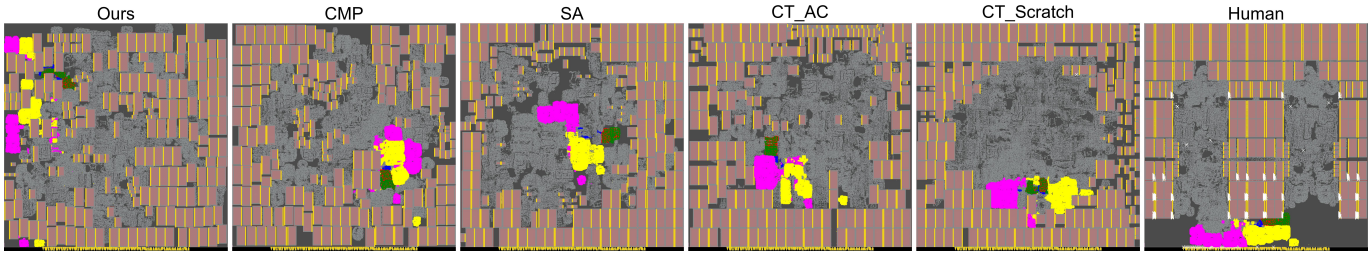

Figure 5: Post-route layouts of bp_quad (NanGate45, 1.3 ns clock) for each macro placement method.

4.1 Experimental Setup

Benchmarks and Flows. For reproducibility, our primary evaluation uses OpenROAD [2] v2.0 (Yosys, RePlAce [5], OpenDP, TritonCTS, FastRoute/TritonRoute, OpenSTA) on all 8 macro-containing nangate45 designs in the OpenROAD flow. To demonstrate flow independence, we evaluate ariane133 (133 macros) and bp_quad (220 macros) on a commercial Cadence flow (Genus 21.18 iSpatial + Innovus 21.19); these are the two TILOS MacroPlacement designs [6] with both RTL and published baselines available. To further demonstrate technology independence, the Cadence evaluation is performed on both nangate45 and ASAP7.

Baselines. For the legalization comparison (RQ1), we evaluate our method against **Greedy** and **LP**. Greedy resolves overlaps by processing macros sequentially, shifting each to the nearest overlap-free position. LP adopts the constraint-graph formulation [7, 19] and minimizes total displacement by linear programming.

For the end-to-end comparison (RQ2), we evaluate our pipeline (RePlAce + our legalization) against several established macro placement methods. On the OpenROAD flow, we compare with **Hier-RTLMP** [9] and **TritonMP**, both are based on simulated annealing (SA). On the Cadence flow, we compare with **CMP** (Cadence's Concurrent Macro Placer), **Circuit Training** (CT [6, 16]; CT_Scratch and CT_AC), **SA**, and **Human expert**, all obtained from the TILOS repository [6].

4.2 Topology Preservation Matters (RQ1)

To answer RQ1, we fix the analytical floorplan from RePlAce and vary only the legalization method. Table 1 reports the post-route PPA of all three methods on 8 OpenROAD nangate45 designs.

Greedy sequentially minimizes per-macro displacement, ignoring the analytical floorplan's topology. Although it often achieves low mean displacement (e.g., 65 μm on ariane133), its scanning order can push individual macros far from their original positions, yielding high maximum displacement (e.g., 145 μm on swerv_wrapper vs. our 26 μm). On black_parrot, the displaced macros cause the downstream flow to fail entirely (Table 1, "N/A"). More importantly, the disrupted topology degrades downstream timing: on ariane133, the largest timing-critical

design, Greedy has the worst WNS (−0.32 ns) despite the lowest mean displacement.

LP largely preserves relative positions through explicit pairwise constraints, achieving competitive WNS and TNS. However, the $O(n^2)$ constraint formulation over-specifies the problem: on six of eight designs, LP requires die expansion of up to 44% to reach a feasible solution. Its timing quality confirms the value of topology preservation, but the die expansion cost limits its practical applicability.

Our method preserves topology and minimizes displacement simultaneously, with no die expansion. On ariane136, for example, we achieve the lowest mean displacement (101 μm) with zero timing violation. These results answer RQ1: a legalizer that preserves both topology and macro positions yields consistent downstream PPA.

4.3 End-to-End Comparison (RQ2)

To answer RQ2, we assess whether analytical floorplanning, paired with our legalization, can compete with established macro placement approaches.

OpenROAD (nangate45). Table 2 compares our pipeline (RePlAce + our legalization) against OpenROAD's two SA-based macro placers: Hier-RTLMP [9] (sequence-pair based) and TritonMP (B*-tree based). Our pipeline achieves the best or tied-best WNS and TNS on 7 of 8 designs, and the best wirelength on 6 of 8 designs.

Cadence Innovus (nangate45 + ASAP7). Table 3 compares our pipeline against a broader set of approaches on a commercial Cadence flow. We run both our pipeline and CMP (marked CMP†) on Innovus 21.19; the remaining baselines use TILOS's Innovus 21.10 results [6]. Our pipeline consistently achieves shorter wirelength than CT, and produces PPA comparable to CMP.

4.4 Analysis of Design Choices (RQ3)

We note that MFTB/TBMF are theoretically grounded. Therefore, we examine the remaining design choices below.

Input Preparation. Table 4 shows that: (1) macro padding makes the analytical floorplan aware of minimum spacing, substantially reducing displacement for both construction methods; (2) the mosaic builder captures topology more accurately than the faster slicing tree builder due to its greater expressiveness. Additionally, bp_quad ASAP7 in Table 3 benefits from per-type inflation of excessively small macros, increasing the shrink ratio from 0.14 to 0.77.

Optimization Convergence. Since macro coordinates are differentiable with respect to filler sizes, gradient descent is a suitable optimization technique. To minimize the dependency on hyperparameter tuning, we use the Adam optimizer [10]. As shown in Fig. 6, a single configuration (learning rate 1.0, gradient clipping at 100, early stopping at 0.1% relative improvement) converges smoothly across all 10 benchmarks.

In conclusion, input preparation is essential for topology quality, and gradient descent is a natural fit and converges well.

4.5 Runtime and Scalability (RQ4)

Fig. 7 compares runtime across both flows. On OpenROAD (a), our legalization completes in 1–59 s, dominated by gradient descent; the structural stages finish in under 1 s. Greedy is surprisingly slow on large designs (600–900 s) because of its exhaustive scanning. LP completes under 0.2 s, but it requires die expansion. On end-to-end OpenROAD (b), our pipeline is 63–74× faster than TritonMP on ariane133/136. On the Cadence flow (c), our pipeline completes in 1–12 minutes on a single CPU, comparable to CMP (3–28 minutes on 8 CPUs). SA requires 10–12 hours on 80 CPUs, and CT requires 36–64 hours on 8 GPUs and 576 CPUs [6].

These results answer RQ4: our method is computationally practical, completing legalization in seconds and the full pipeline in minutes on a single CPU. Each iteration runs in $O(n)$ time, and gradient descent can be further accelerated with GPU support.

Table 4: Effect of construction method and macro padding. Mean, Max, Med = ℓ^2 displacement (μm); Viol = boundary violations.

Configuration (ariane133)	Mean	Max	Med.	Viol.
Slicing tree builder	78.6	311.9	58.2	11
Slicing tree builder + macro padding	66.7	269.2	47.4	11
Mosaic builder	76.8	179.9	77.1	16
Mosaic builder + macro padding	**12.7**	**80.9**	**10.2**	**5**

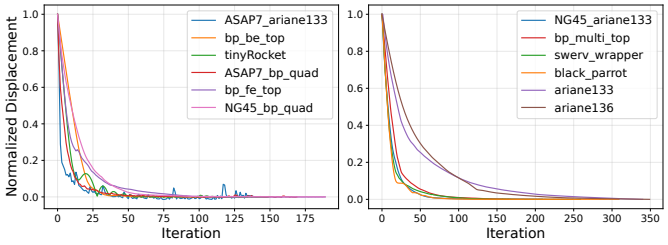

Figure 6: Normalized displacement convergence across 10 designs. Designs prefixed with technology are evaluated on Cadence.

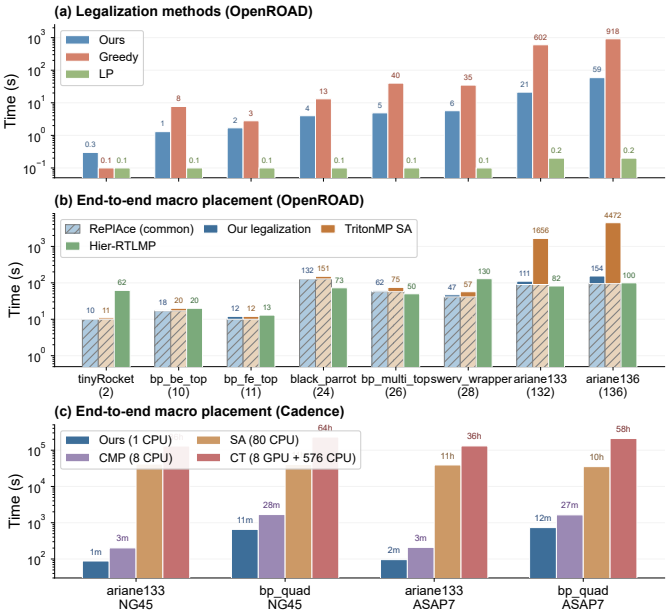

Figure 7: Runtime comparison. (a) Legalization only (OpenROAD). (b) End-to-end (OpenROAD); both our legalization method and TritonMP SA method contain the runtime of RePlAce (hatched area). (c) End-to-end (Cadence): resources are shown in the legend.

5 Conclusion

In this paper, we present a new legalization framework that simultaneously minimizes macro displacements and preserves topology from a given analytical floorplan. We encode the floorplan as twin binary trees, so that block coordinates are piecewise linear functions of filler sizes. The optimization problem thus becomes differentiable, and no overlaps arise between blocks throughout the process. Our experiments on industrial benchmarks confirm that the PPA results of our method are comparable to those produced by commercial tools. Our method can be adapted to other tasks in VLSI physical design, such as layout optimization with a generative model or RL agent. We leave these directions for our future work.

References

[1] Anthony Agnesina, Puranjay Rajvanshi, Tian Yang, Geraldo Pradipta, Austin Jiao, Ben Keller, Brucek Khailany, and Haoxing Ren. 2023. AutoDMP: Automated DREAMPlace-based Macro Placement. In *Proceedings of the 2023 International Symposium on Physical Design*. 149–157.

[2] T Ajayi, D Blaauw, TB Chan, CK Cheng, VA Chhabria, DK Choo, M Coltella, S Dobre, R Dreslinski, M Fogaça, et al. 2019. OpenROAD: Toward a Self-Driving, Open-Source Digital Layout Implementation Tool Chain. *Proc. GOMACTECH* (2019), 1105–1110.

[3] Yun-Chih Chang, Yao-Wen Chang, Guang-Ming Wu, and Shu-Wei Wu. 2000. B*-trees: A new representation for non-slicing floorplans. In *Proceedings of the 37th annual design automation conference*. 458–463.

[4] Yifan Chen, Zaiwen Wen, Yun Liang, and Yibo Lin. 2023. Stronger mixed-size placement backbone considering second-order information. In *2023 IEEE/ACM International Conference on Computer Aided Design (ICCAD)*. IEEE, 1–9.

[5] Chung-Kuan Cheng, Andrew B Kahng, Ilgweon Kang, and Lutong Wang. 2019. RePlAce: Advancing Solution Quality and Routability Validation in Global Placement. *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems* 38, 9 (2019), 1717–1730.

[6] Chung-Kuan Cheng, Andrew B Kahng, Sayak Kundu, Yucheng Wang, and Zhiang Wang. 2023. Assessment of reinforcement learning for macro placement. In *Proceedings of the 2023 International Symposium on Physical Design*. 158–166.

[7] Jason Cong and Min Xie. 2008. A robust mixed-size legalization and detailed placement algorithm. *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems* 27, 8 (2008), 1349–1362.

[8] Dwight Hill. 2002. Method and system for high speed detailed placement of cells within an integrated circuit design. US Patent 6,370,673.

[9] Andrew B Kahng, Ravi Varadarajan, and Zhiang Wang. 2023. Hier-RTLMP: A hierarchical automatic macro placer for large-scale complex IP blocks. *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems* 43, 5 (2023), 1552–1565.

[10] Diederik P Kingma and Jimmy Ba. 2014. Adam: A method for stochastic optimization. *arXiv preprint arXiv:1412.6980* (2014).

[11] Yao Lai, Yao Mu, and Ping Luo. 2022. MaskPlace: Fast chip placement via reinforced visual representation learning. *Advances in Neural Information Processing Systems* 35 (2022), 24019–24030.

[12] Peiyu Liao, Dawei Guo, Zizheng Guo, Siting Liu, Yibo Lin, and Bei Yu. 2023. DREAMPlace 4.0: Timing-driven placement with momentum-based net weighting and lagrangian-based refinement. *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems* 42, 10 (2023), 3374–3387.

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

[14] Lixin Liu, Bangqi Fu, Shiju Lin, Jinwei Liu, Evangeline FY Young, and Martin DF Wong. 2024. Xplace: An Extremely Fast and Extensible Placement Framework. *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems* 43, 6 (2024), 1872–1885.

[15] Jingwei Lu, Hao Zhuang, Pengwen Chen, Hongliang Chang, Chin-Chih Chang, Yiu-Chung Wong, Lu Sha, Dennis Huang, Yufeng Luo, Chin-Chi Teng, et al. 2015. ePlace-MS: Electrostatics-based placement for mixed-size circuits. *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems* 34, 5 (2015), 685–698.

[16] Azalia Mirhoseini, Anna Goldie, Mustafa Yazgan, Joe Wenjie Jiang, Ebrahim Songhori, Shen Wang, Young-Joon Lee, Eric Johnson, Omkar Pathak, Azade Nova, et al. 2021. A graph placement methodology for fast chip design. *Nature* 594, 7862 (2021), 207–212.

[17] Hiroshi Murata, Kunihiro Fujiyoshi, Shigetoshi Nakatake, and Yoji Kajitani. 2002. VLSI module placement based on rectangle-packing by the sequence-pair. *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems* 15, 12 (2002), 1518–1524.

[18] John K Ousterhout. 1984. Corner stitching: A data-structuring technique for VLSI layout tools. *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems* 3, 1 (1984), 87–100.

[19] Yuan Pu, Tinghuan Chen, Zhuolun He, Jiajun Qin, Chen Bai, Haisheng Zheng, Yibo Lin, and Bei Yu. 2025. IncreMacro: Incremental Macro Placement Refinement. *IEEE TRANSACTIONS ON COMPUTER-AIDED DESIGN OF INTEGRATED CIRCUITS AND SYSTEMS* 44, 8 (2025).

[20] Yunqi Shi, Ke Xue, Song Lei, and Chao Qian. 2023. Macro placement by wire-mask-guided black-box optimization. *Advances in Neural Information Processing Systems* 36 (2023), 6825–6843.

[21] Bo Yao, Hongyu Chen, Chung-Kuan Cheng, and Ronald Graham. 2003. Floorplan representations: Complexity and connections. *ACM Transactions on Design Automation of Electronic Systems (TODAES)* 8, 1 (2003), 55–80.

[22] Hai Zhou and Jia Wang. 2004. ACG-adjacent constraint graph for general floorplans. In *IEEE International Conference on Computer Design: VLSI in Computers and Processors, 2004. ICCD 2004. Proceedings.* IEEE, 572–575.