E-Layout: Layout-Stage Logic Restructuring via RL-Optimized Equality-Saturated Graph Retrieval-Augmented Generation

Wenji Fang[1†], Rongjian Liang[2], Yi-Chen Lu[3], Cunxi Yu[2,4],
Chen Chen[4], Shang Liu[1], Chenhui Deng[2], Wenkai Li[1], Haoxing Ren[5], Zhiyao Xie[1*]
[1]Hong Kong University of Science and Technology
[2]NVIDIA, [3]Ricursive Intelligence, [4]University of Maryland, College Park, [5]Agentrys,

ABSTRACT

Achieving high sign-off performance in advanced ICs increasingly depends on layout-stage optimization under realistic physical conditions. While recent efforts have moved toward synthesis–layout fusion, most existing works still focus on bringing estimated physical information earlier into synthesis, whose optimization does not fully correlate with real layout-stage behavior. Meanwhile, layout-stage optimization remains largely restricted to a fixed logic structure, relying mainly on buffering and sizing. This fundamentally constrains the optimization space under realistic physical conditions. To address this, we present E-Layout, an ML-driven framework for post-placement layout restructuring. E-Layout jointly optimizes logic transformation and physical implementation across multiple subcircuits under design-level timing feedback. This enables the selection of the most timing-beneficial set of disjoint restructured subcircuits to form the optimized circuit. Specifically, it first constructs equivalence-preserving logic alternatives via e-graph equality saturation, then couples logic selection with physical realization through a graph-based retrieval-augmented generation (RAG) model, and finally refines these local logic–physical decisions with reinforcement learning under design-level timing feedback. Evaluated on 9 diverse designs, it achieves on average 6% TNS improvement and 2% WNS improvement, while even reducing area by 1%, over an industry-leading commercial physical design flow.

1 INTRODUCTION

Modern VLSI design flows have traditionally treated logic synthesis and physical design as largely separate stages. Logic synthesis determines most of the gate-level structure early, while downstream layout stages primarily refine the physical implementation through placement, routing, and related optimizations, such as buffering and gate sizing. Although this separation simplifies flow organization and EDA tool development, it also fixes many logic structural decisions before accurate physical information becomes available. As a result, optimization opportunities that depend on real placement and interconnect context are often difficult to fully exploit in later physical stages.

The trend of synthesis–layout fusion. In recent years, this rigid boundary has gradually blurred. In industry, commercial EDA tools increasingly emphasize tighter cross-stage integration to improve optimization at advanced technology nodes [8, 39]. In academia, this trend has been explored mainly through physical-aware synthesis [19, 27, 29, 36], where estimated placement and parasitic information are fed back into earlier synthesis stages to guide iterative resynthesis. Together, these efforts reflect a broader shift from stage-isolated optimization toward logic–physical co-optimization.

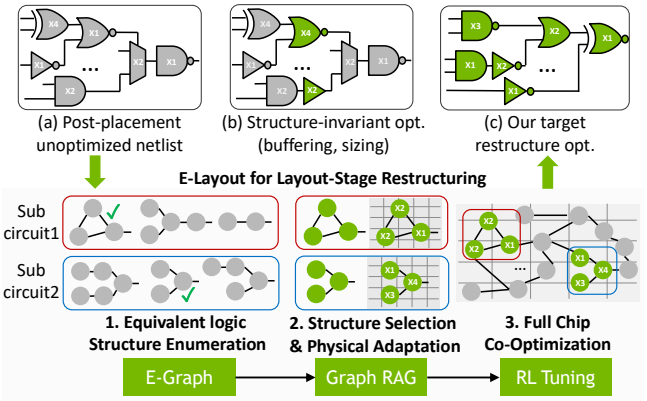


Figure 1: Overview of E-Layout. E-Layout targets layout-stage restructuring, opening a much larger optimization space than structure-invariant methods. It decomposes the task into three coupled steps: equivalent structure exploration, logic selection with physical realization, and design-level timing-driven refinement across disjoint subcircuits.

The current academic focus: physical-aware synthesis. Despite this broader trend, most recent academic efforts have focused on bringing physical information earlier into synthesis, rather than enabling direct restructuring later at the layout stage. This distinction is important: physical-aware synthesis operates before final placement is available and therefore relies on approximate physical estimates. While such estimates can guide early logic optimization, they do not fully capture the true post-placement timing behavior of the final implementation, and may therefore lead to restructuring decisions that deviate from the actual optimum.

Layout-stage optimization. This limitation naturally shifts the focus toward the layout stage, where direct restructuring can be performed under the real physical context, as shown in Figure 1. At this stage, gate locations and interconnect context are available, enabling more accurate timing correlation. However, optimization at this stage remains dominated by *structure-invariant* techniques, such as buffering [4, 15, 22, 24, 26, 32, 41, 42] and gate sizing [13, 31, 33, 34, 44, 47], which refine the physical implementation without changing the underlying logic structure. This fundamentally limits the optimization space at the layout stage.

[†]Work done while Wenji Fang was an intern at NVIDIA.
[*]Corresponding author (eezhiyao@ust.hk).

ICCAD '26, November 08–12, 2026, San Jose, CA, USA

ACM ISBN 979-8-4007-2873-0/2026/11
https://doi.org/10.1145/3831252.3833976

Table 1: Comparison with existing layout-stage logic restructuring methods.

Feature		OpenROAD Restructure [3]	OpenPhySyn Restructure [2]	E-Layout (ours)
Rewrite	Method	re-mapping	pin-swap, gate clone	e-graph + graph RAG
	Scope	subcircuit	cell	subcircuit
	Logic change	✓	✗	✓
Opt Impact	Design timing	✗	✓	✓
	Delay target	ABC (unit delay)	layout (NLDM)	layout (NLDM)
	Opt. space	large	small	medium

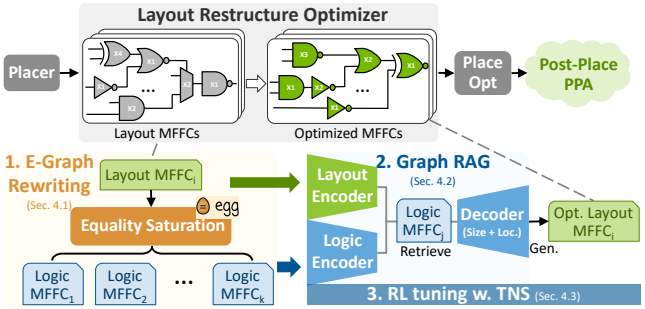


Figure 2: Workflow of E-Layout and its integration into the placement flow. E-Layout first uses e-graph equality saturation to provide functionally equivalent logic MFFC candidates, then applies an RL-optimized graph RAG model to select new structures and generate gate sizes and locations.

Limitations of existing layout restructuring. To enlarge the optimization space beyond structure-invariant buffering and sizing, a limited number of earlier works [9, 23, 25, 37] also explored layout-stage restructuring decades ago. These studies investigated layout-driven restructuring through placement-aware decomposition, incremental rewiring, and layout-level remapping. However, they remained largely heuristic and locally scoped: the restructuring space was limited to narrow rewiring or remapping opportunities on local subcircuits, and physical realization was not jointly optimized with logic transformation.

This limitation also persists in modern, widely adopted open-source tools. For example, OpenROAD [3] identifies timing-critical regions and invokes ABC [7] for logic remapping. However, because ABC operates purely at the logic level without physical awareness, the resulting remapping can significantly break physical coherence and lead to weak post-placement correlation. OpenPhySyn [2] performs physically aware local edits such as pin swapping and gate cloning, together with timing-repair operations such as buffer insertion and resizing, but these cell-level transforms offer only limited restructuring capability and cannot explore meaningfully different logic structures. Consequently, modern open-source layout-stage restructuring remains either loosely coupled to physical context or restricted to narrow local edits.

Our solution: post-placement logic–layout co-optimization. To address these limitations, we propose **E-Layout**, an ML-driven framework for post-placement[1] layout restructuring, as shown in Figure 2. The key idea is to jointly optimize logic structure transformation and physical adaptation (i.e., resize and relocate for each new cell) across multiple subcircuits under global design-level timing feedback. As a result, the most timing-beneficial set of disjoint restructurings can be selected to form the optimized circuit.

Specifically, E-Layout operates at the subcircuit level to balance rich restructuring opportunities with physically coherent local updates, while jointly optimizing multiple local subcircuits under global design timing feedback. It decomposes layout restructuring into three steps, each enabled by a dedicated strategy:

(1) **Equivalent structure exploration.** Layout-stage restructuring should rewrite the post-synthesis logic structure while strictly preserving functional equivalence. To achieve this, E-Layout uses e-graph equality saturation to expand each local subcircuit into a compact space of functionally equivalent pure logic structures. This enables much broader structural exploration than isolated local rewrites while preserving correctness by construction.

(2) **Logic selection with physical realization.** A promising logic structure is useful only after it is physically realized and evaluated under the surrounding layout context. To this end, we develop a graph retrieval-augmented generation (RAG) model to retrieve promising structures from the e-graph rewritten candidate space and jointly generate physically consistent gate sizes and locations, thereby coupling logic selection with physical adaptation.

(3) **Design-level timing refinement.** Promising local subcircuit restructurings should not be judged in isolation, because their timing impact depends on full-chip interaction across all subcircuits in the design. To this end, E-Layout further tunes the graph RAG model with reinforcement learning (RL) using full-chip timing feedback, refining local logic–physical decisions toward improved design-level post-placement timing.

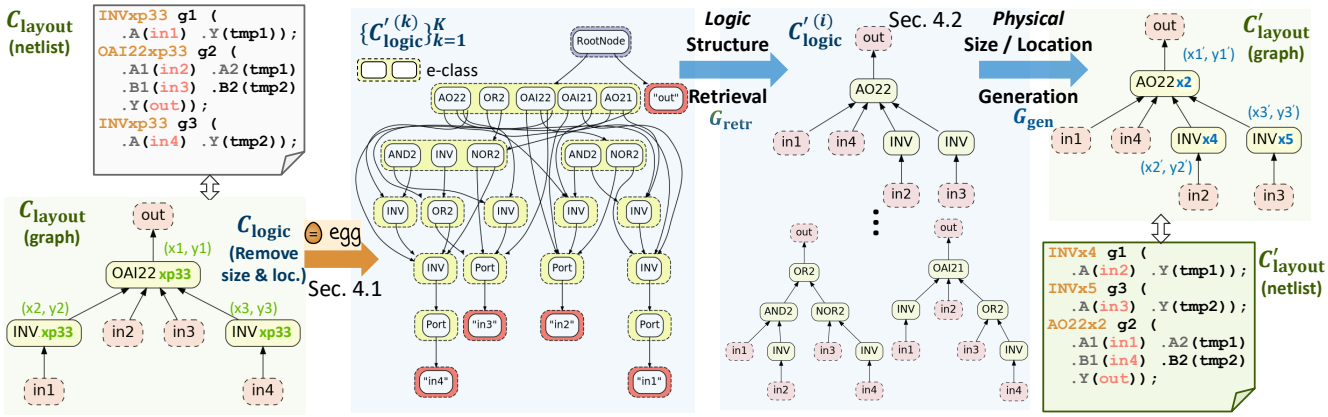
Evaluated on 9 diverse designs, E-Layout delivers on average 6% TNS improvement and 2% WNS improvement, while even achieving 1% area reduction over a commercial physical design flow that has already undergone extensive optimization. We further demonstrate its impact on critical paths at both the design and subcircuit levels. E-Layout is also designed for scalability: e-graph rewriting is applied to non-overlapping subcircuits that can be processed in parallel, while graph RAG inference in each RL iteration completes within seconds. Together, these results highlight the promise of combining symbolic reasoning with ML techniques for layout-stage restructuring optimization.

2 PRELIMINARIES

2.1 MFFC in Circuit Optimization

A *maximum fanout-free cone* (MFFC) is a common structural abstraction in logic synthesis, including both logic rewriting and technology mapping [6, 12]. Formally, the MFFC of a node is the largest subset of nodes such that every path from any node in the subset to the primary outputs passes through that node. Intuitively, it contains all logic used exclusively to drive that node, and can therefore be replaced as a whole when the node is substituted. This makes MFFCs a natural unit for local circuit optimization, since transformations within an MFFC do not directly affect other parts

[1]We choose post-placement because it is the earliest stage with real physical information. Earlier synthesis stages rely on estimates, whereas later sign-off and ECO stages typically offer a smaller optimization space. In principle, the framework can also be extended to other stages.



(a) Layout MFFC netlist and graph (b) Equality-saturated logic MFFCs (c) Retrieved logic MFFCs (d) Optimized layout MFFC

Figure 3: E-Layout restructuring flow in the MFFC perspective. (a) Extract a post-placement layout MFFC graph. (b) Apply e-graph equality saturation to enumerate equivalent logic structures. (c) Retrieve logic structure candidates, then (d) Generate physical sizes and locations to form an optimized layout MFFC to substitute the original one.

of the circuit through shared internal logic. In E-Layout, we likewise adopt MFFCs as the basic subcircuit granularity for layout restructuring.

2.2 E-Graphs for Circuit Optimization

An e-graph is a compact data structure for representing many equivalent expressions simultaneously, and has been increasingly adopted in circuit optimization. E-graphs are typically used in a two-stage workflow: *equality saturation* followed by *extraction*. During equality saturation, rewriting rules are repeatedly applied to expand the e-graph with functionally equivalent circuit structures. Once saturation reaches a predefined resource limit or fixed point, an optimized implementation is extracted according to a target cost function, such as area or delay. By avoiding early commitment to a single rewriting sequence, e-graphs enable efficient exploration of large equivalence spaces and have been applied to a variety of circuit representations, including RTL [17, 18, 28, 45], AIGs [10, 11], and HLS [14, 40]. Despite their success in logic optimization, standard e-graphs are designed for pure logic representations and therefore do not directly support layout-stage restructuring with physical attributes and post-placement timing.

3 PROBLEM FORMULATION

Given a placed but unoptimized layout $\mathcal{G}$, our goal is to perform post-placement restructuring and obtain a restructured layout $\mathcal{G}'$ that improves timing while preserving functional equivalence. After restructuring, $\mathcal{G}'$ is passed back to the commercial physical design tool for incremental legalization and subsequent structure-invariant optimizations such as buffering and sizing. The final quality of the restructured design is then evaluated using timing and area metrics, including total negative slack (TNS), worst negative slack (WNS), and area. Formally, we seek

$$\mathcal{G}' = \arg \min_{\hat{\mathcal{G}} \in \Omega(\mathcal{G})} \mathcal{L}_{\text{PPA}}(\hat{\mathcal{G}}) \quad \text{s.t.} \quad \hat{\mathcal{G}} \equiv \mathcal{G},$$

where $\Omega(\mathcal{G})$ denotes the set of physically realizable restructured layouts derived from $\mathcal{G}$ through layout-stage restructuring, $\hat{\mathcal{G}} \equiv \mathcal{G}$ enforces functional equivalence, and $\mathcal{L}_{\text{PPA}}$ is an optimization objective defined over post-layout metrics, including TNS, WNS, and area.

Specifically, layout-stage restructuring should jointly optimize two coupled aspects: (1) *logic transformation*, by replacing local subcircuits with functionally equivalent alternatives; and (2) *physical adaptation*, by assigning physically consistent attributes to the transformed structure, including gate sizes and placement locations of each cell.

4 METHODOLOGY

The overall framework of E-Layout is shown in Figure 2. E-Layout is built on the view that layout-stage restructuring should couple *logic transformation* and *physical implementation*, and optimize them under *design-level timing feedback*. Accordingly, it proceeds in three tightly connected stages. First, as described in Section 4.1, E-Layout applies e-graph rewriting to layout standard-cell graphs to construct an equivalence-preserving space of candidate logic structures. Second, as illustrated in Section 4.2, it uses a graph RAG model to select promising structures from these alternatives and generate their physical sizes and locations, thereby bridging logic and physical implementation. Third, as detailed in Section 4.3, it further refines these local logic–physical decisions by tuning the graph RAG model with reinforcement learning under layout timing feedback, so that restructuring is ultimately guided by full-chip timing improvement.

4.1 Layout MFFC Rewriting via E-Graph Equality Saturation

At the first stage of E-Layout, we use e-graph rewriting to expand the structural search space by systematically enumerating functionally equivalent alternatives on layout standard-cell graphs. In this way, E-Layout exposes richer restructuring opportunities while preserving correctness.

However, directly applying the standard e-graph saturation and extraction paradigm to our target layout is not applicable, mainly due to two challenges:

Table 2: Standard cell rewriting rules.

Type	Standard Cell Rewriting Rules	
Type I: one-level de/recompose for complex gates	$\text{AO22} \Leftrightarrow \text{AO21}(x_1, x_2, \text{AND2}(y_1, y_2))$ $\text{AND3} \Leftrightarrow \text{AND2}(\text{AND2}(x, y), z)$ $\text{OA22} \Leftrightarrow \text{INV}(\text{AO21}...)$ …	
Type II: complex ⇔ primitive	$\text{AO22} \Leftrightarrow \text{OR2}(\text{AND2}(x1, x2), \text{AND2}(y1, y2))$ $\text{NAND2} \Leftrightarrow \text{INV}(\text{AND2}(x, y))$ $\text{XOR2} \Leftrightarrow \text{OR2}(\text{AND2}(x, \text{INV}(y)), \text{AND2}(\text{INV}(x), y))$ …	
Type III: Boolean simplification	*De Morgan*	$\text{INV}(\text{AND2}(x, y)) \Leftrightarrow \text{OR2}(\text{INV}(x), \text{INV}(y))$
	Double inversion	$\text{INV}(\text{INV}(x)) \Rightarrow x$
	Absorption	$\text{AND2}(x, \text{OR2}(x, y)) \Rightarrow x$
	…	

- **Saturation: difficult on complex standard cells with physical attributes.** Layout circuits contain diverse standard cells with complex logic functions and physical attributes. Existing e-graph rewriting rule sets are mainly designed for pure and simple logic forms (e.g., AIG and RTL), making them hard to reuse for standard-cell graphs.
- **Extraction: lack of PPA-aware cost.** Conventional e-graph extraction is driven by simple linear cost functions (e.g., node count, logic depth), which cannot capture layout PPA with non-linear, topology- and placement-dependent delays, thus limiting physical-aware logic optimization.

As shown in Figure 3, to overcome these challenges, we leverage the e-graph only to provide diverse equivalent pure-logic structures via saturation. Then, the extraction of physically optimized layouts, which selects suitable structures and assigns new gate physical attributes, is delegated to our subsequent graph-based RAG model. We detail the e-graph equality saturation process designed for standard-cell layout below.

Handling layout at MFFC level. As shown in Figure 3 (a), we perform e-graph saturation at the MFFC granularity, denoted as C_{layout}. Since no logic inside an MFFC is shared with nodes outside the cone, optimizing an MFFC can directly improve PPA without disturbing the rest of the circuit [12]. This allows us to explore new structures for multiple nodes within an MFFC independently. Specifically, we extract the MFFC for each gate from the placed, unoptimized layout and represent it as a graph. To enable pure-logic rewriting, we remove all physical attributes (i.e., size, location) from C_{layout} and retain only the logical structure for e-graph saturation, denoted as C_{logic}.

E-graph saturation with layout rewriting rules. To support e-graph rewriting over standard-cell layouts, we design a compact set of equivalence-preserving rules over library cells, as summarized in Table 2. These rules expand the search space through decomposition, recomposition, and Boolean simplification. Specifically, we propose three types of standard cell rewriting rules: (1) one-level decomposition/recomposition of complex gates, where complex cells (e.g., AO22) are decomposed into smaller cells (e.g., AO21 + AND2) and recomposed back; (2) complex-to-primitive bidirectional conversions, where complex cells (e.g., AO22) are transformed into multiple primitive gates (e.g., AND2, OR2, INV) and vice versa, enabling finer-grained restructuring; and (3) Boolean simplification, where canonical identities (e.g., De Morgan, double inversion removal, absorption) are applied to provide simplified implementations.

With these customized rules, we construct an initial e-graph from each layout MFFC C_{layout} and run equality saturation until reaching resource limits (e.g., iteration bound, e-node budget, or fixpoint) to enumerate all K equivalent logic MFFCs $\{C'^{(k)}_{\text{logic}}\}_{k=1}^{K}$. These equivalent logic MFFCs serve the candidate pool for our subsequent ML framework. During saturation, the rules systematically enumerate alternative implementations while guaranteeing functional equivalence. As shown in Figure 3 (b), the resulting saturated e-graph compactly contains a rich design space of candidate pure-logic structures for each MFFC, which will later serve as input to our graph-based RAG model for timing optimization.

4.2 Graph RAG Model Architecture

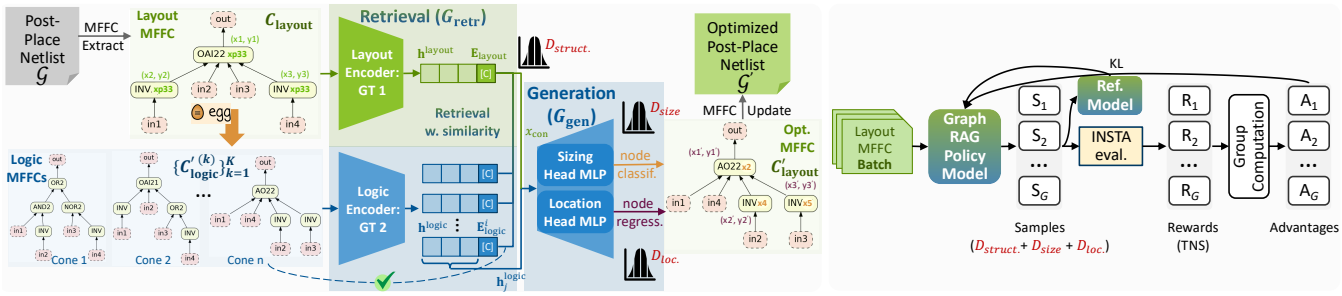
While equality saturation provides a rich space of equivalent MFFCs, selecting a timing-optimal structure cannot be reliably achieved with simple linear extraction costs. To address this, we use a graph-based retrieval-augmented generation (RAG) model to connect structural exploration with physical realization. The equivalent pure-logic candidates $\{C'^{(k)}_{\text{logic}}\}_{k=1}^{K}$ serve as the retrieval space, from which the promising structure C'_{logic} is selected, while gate sizes and locations are generated conditioned on the selected structure and the surrounding layout context. This allows E-Layout to transform equivalence-preserving logic candidates into physically optimized layout subcircuits. As shown in Figure 3(c) and (d), the retrieved logic structure C'_{logic}, together with the predicted sizes and locations, forms an optimized C'_{layout} that replaces the original C_{layout}.

As shown in Figure 4(a), the graph RAG model consists of two components: (1) a retrieval module G_{retr}, which uses separate graph encoders for the original layout MFFC and the candidate logic MFFCs to retrieve promising structures; and (2) a generation module G_{gen}, which takes the retrieved logic structure together with the original layout context and predicts per-cell sizes and locations. We detail each module below.

Logic structure retrieval: dual-tower layout-logic graph encoders. The retrieval module G_{retr} contains a layout encoder and a logic encoder. It encodes the original layout MFFC C_{layout} and the candidate logic MFFCs $\{C'_{\text{logic}}(k)\}_{k=1}^{K}$ into layout embedding $\mathbf{E}_{\text{layout}}$ and logic embedding pool $\{\mathbf{E}^{(k)}_{\text{logic}}\}_{k=1}^{K}$, respectively. Then $\mathbf{E}_{\text{layout}}$ serves as the query to retrieve candidates from the logic pool $\{\mathbf{E}^{(k)}_{\text{logic}}\}_{k=1}^{K}$ via similarity scoring.

Specifically, the layout encoder maps C_{layout} into an embedding using both physical and structural information. Given the layout MFFC graph, for each node n_i it combines physical features $\mathbf{x}_i^{\text{phys.}}$ (e.g., cell size, capacitance, slew, slack, and area) and structural features $\mathbf{x}_i^{\text{struct.}}$ (e.g., cell type and number of reachable endpoints) to produce a node embedding $\mathbf{h}_i^{\text{layout}}$. These node embeddings are then aggregated over all nodes to obtain a single cone-level embedding $\mathbf{E}_{\text{layout}}$. The logic encoder maps C'_{logic} using structural information only. It operates on equality-saturated candidate MFFCs and uses logical features (e.g., cell type and fanout-of-4 delay) to produce node embeddings $\mathbf{h}_j^{\text{logic}}$, which are similarly aggregated into a graph-level embedding $\mathbf{E}_{\text{logic}}$. Both encoders are implemented as graph transformers followed by pooling to obtain the graph-level embedding, formulated as

$$G_{\text{retr}}: \begin{cases} \mathbf{h}_i^{\text{layout}} = \text{GT}_1(\mathbf{x}_i^{\text{phys.}} + \mathbf{x}_i^{\text{struct.}}), \mathbf{E}_{\text{layout}} = \text{Pool}\Big(\{\mathbf{h}_i^{\text{layout}}\}_i\Big) \\ \mathbf{h}_j^{\text{logic}} = \text{GT}_2(\mathbf{x}_j^{\text{struct.}}), \mathbf{E}_{\text{logic}} = \text{Pool}\Big(\{\mathbf{h}_j^{\text{logic}}\}_j\Big) \end{cases} \tag{1}$$



(a) Graph retrieval-augmented generation (RAG) model architecture (b) Graph RAG RL-tuning with TNS feedback

Figure 4: Graph RAG for logic structure retrieval and physical attribute generation, with RL tuning with layout TNS feedback.

During retrieval, we compute the cosine similarity between the layout embedding query $\mathbf{E}_{\text{layout}}$ and each logic embedding $\mathbf{E}_{\text{logic}}^{(k)}$ from the embedding pool to rank the equivalent logic structures:

$$\text{Retrieve: } \text{Sim}_{\cos}(\mathbf{E}_{\text{layout}}, \mathbf{E}_{\text{logic}}^{(k)}) = \frac{\mathbf{E}_{\text{layout}} \cdot \mathbf{E}_{\text{logic}}^{(k)}}{\|\mathbf{E}_{\text{layout}}\|_2 \, \|\mathbf{E}_{\text{logic}}^{(k)}\|_2} \tag{2}$$

Physical attribute generation: cell-level decoder. After retrieving the new logic structures, we generate their cell-level physical attributes using two decoder heads. For each gate, the decoder takes a combined feature $\mathbf{h}_j^{\text{dec}}$ as input. It consists of 1) the retrieved logic embedding $\mathbf{h}_j^{\text{logic}}$, 2) the layout MFFC constraints $\mathbf{x}_{\text{con}}$ (i.e., its IO net slack and slew), and 3) the layout MFFC embedding $\mathbf{E}_{\text{layout}}$. This combined feature conditions the cell decoder to generate sizes and locations that remain consistent with the surrounding physical context, ensuring that the rewritten structure integrates smoothly into the existing layout.

The decoder uses two MLP heads taking $\mathbf{h}_j^{\text{dec}}$ as input for sizing classification and location regression, respectively. The sizing head MLP_{size} performs masked classification. It predicts logits over all size options and applies a legality mask based on each cell type so that final predictions are restricted to library-valid sizes only. The location head MLP_{loc} performs normalized regression. It predicts the normalized coordinates $(\tilde{x}_j, \tilde{y}_j) \in [-1, 1]^2$ for each gate, which are linearly mapped to the original MFFC bounding box to obtain physical coordinates, ensuring all placements remain within a feasible region while still allowing flexible adjustments. The generation process is formulated as follows:

$$G_{\text{gen}}: \begin{cases} \mathbf{h}_j^{\text{dec}} = \left[\mathbf{h}_j^{\text{logic}}, \, \mathbf{x}_{\text{con}}, \, \mathbf{E}_{\text{layout}}\right] & \text{(Gate feat.)} \\ p(s_j \mid \mathbf{h}_j^{\text{dec}}) = \text{Softmax}\Big(\text{MLP}_{\text{size}}\Big(\text{mask}(\mathbf{h}_j^{\text{dec}})\Big)\Big) & \text{(Size)} \\ (\tilde{x}_j, \tilde{y}_j) = \text{MLP}_{\text{loc}}\Big(\mathbf{h}_j^{\text{dec}}\Big), \quad (\tilde{x}_j, \tilde{y}_j) \in [-1, 1]^2 & \text{(Location)} \end{cases} \tag{3}$$

This encoder–decoder graph RAG model bridges logic structure and physical attributes, enabling each layout MFFC to be rewritten into a new variant that can be accurately evaluated under layout timing. Moreover, the RAG model can be trained in a data-driven manner to jointly select alternative logic structures and generate corresponding cell sizes and locations, paving the way for our next step of RL-based timing optimization.

4.3 Tuning Graph RAG with RL for Global Timing Optimization

The key challenge is that layout-stage restructuring should be optimized for *design-level timing*, yet no direct supervision exists for this objective. There is no practical ground-truth label for the best logic structure, gate sizes, and gate locations of each local subcircuit once full-chip timing interaction is taken into account. Existing tool solutions are also largely local and heuristic, and therefore do not provide suitable supervision for globally optimal restructuring.

We therefore further optimize the graph RAG model with reinforcement learning using full-chip TNS as reward feedback. This allows E-Layout to refine logic structure selection and physical realization across multiple MFFCs according to their actual design-level timing impact. In this way, RL provides the link between local restructuring decisions and the global post-placement timing objective.

GRPO sampling and reward design. We adopt Group Relative Policy Optimization (GRPO) [21, 38], a lightweight policy-gradient reinforcement learning method. Its key idea is to sample multiple candidate actions for the same state, evaluate them with a task-level reward, and update the ML model according to their *relative* performance within the group. In our setting, the graph RAG model refers to the policy, and layout timing improvement defines the reward. For each layout MFFC, denoted as state s, we sample a group of G candidate restructurings. Each candidate is a joint action $a^{(g)}$ drawn from three model distributions: the structure-retrieval distribution $D_{\text{struct.}}$ from G_{retr}, and the sizing and location distributions D_{size} and $D_{\text{loc.}}$ from G_{gen}.

The sampled candidates replace the original layout MFFC to form G new layouts, which are evaluated by the fast GPU-accelerated incremental STA engine INSTA [30]. To capture interactions among different MFFCs under full-chip timing, we sample and update candidates over a mini-batch of MFFCs. Based on their timing outcomes, we then compute group-wise relative advantages as:

$$A^{(g)} = \frac{\Delta\text{TNS}^{(g)} - \mu_{\text{TNS}}}{\sigma_{\text{TNS}}}, \tag{4}$$

where ΔTNS denotes the improvement in total negative slack. Each objective is standardized independently to maintain scale invariance. GRPO uses the mean advantage within each candidate group as an implicit baseline, eliminating the need for an explicit value-function critic or external ranking model and thus reducing computational overhead.

GRPO training iteration. During GRPO training, we maintain a stochastic graph RAG model π_θ and a frozen reference graph RAG

Table 3: Comparison of post-placement optimization PPA results. E-Layout outperforms state-of-the-art layout restructuring tools, delivering larger timing improvements.

Design	Commercial Tool			w. OpenROAD-Restructure [3]			w. OpenPhySyn-Restructure [2]			w. E-Layout (ours)		
	TNS (ns)	WNS (ns)	Area (mm^2)	TNS (ns)	WNS (ns)	Area (mm^2)	TNS (ns)	WNS (ns)	Area (mm^2)	TNS (ns)	WNS (ns)	Area (mm^2)
s838_1_bench	-1.93	-0.108	46	-3.1 (60.62%)	-0.184	41	-1.891 (-2.02%)	-0.102	45	-1.99 (3.02%)	-0.11 (1.82%)	45
wb_dma_top	-47.76	-0.103	426	-67 (40.28%)	-0.158	386	-45.5 (-4.73%)	-0.11	424	-42.33 (-12.83%)	-0.101 (-1.98%)	372
spi_top	-31.92	-0.192	259	-31.2 (-2.26%)	-0.154	240	-31.3 (-1.94%)	-0.192	267	-30.861 (-3.43%)	-0.191 (-0.52%)	268
usbf_top	-13.1	-0.126	1070	N/A⋆						-11.268 (-16.26%)	-0.108 (-16.67%)	1041
NV_csb_master	-77.1	-0.1	1253	-72 (-6.61%)	-0.117	1235	-73 (-5.32%)	-0.093	1253	-70.32 (-9.6%)	-0.105 (4.76%)	1282
NV_bdma	-58.1	-0.111	539	N/A⋆						-59 (1.53%)	-0.112 (0.89%)	547
ac97_top	-92.5	-0.085	1183	-107 (15.68%)	-0.107	1181	-94 (1.62%)	-0.092	1177	-83.1 (-11.31%)	-0.085 (0%)	1193
arm_core	-558	-0.594	2196	N/A⋆						-550 (-1.45%)	-0.564 (-5.32%)	2271
Rocket	-555	-0.405	1378							-537.6 (-3.24%)	-0.391 (-3.58%)	1360
Avg.				**21.54%**	**29.37%**	**-5.82%**	**-1.37%**	**0.50%**	**0%**	**-6.07%**	**-2.29%**	**-1.07%**

⋆ N/A indicates cases where the corresponding open-source flow cannot support multi-output gates, so the design cannot be processed under that baseline.

model π_{ref}. In each iteration, we sample a mini-batch of MFFCs and use π_θ to generate a group of candidate restructuring solutions for each MFFC, as described in the previous paragraph. These candidates are substituted back into the layout and evaluated by INSTA to obtain timing slack improvement ΔTNS, which is then normalized to compute the group-relative advantage $A^{(g)}$. Finally, we update π_θ by maximizing a clipped surrogate objective:

$$J_{\text{GRPO}}(\theta) = \mathbb{E}\Big[\min\big(r^{(g)}A^{(g)},\ \text{clip}(r^{(g)}, 1-\epsilon, 1+\epsilon)A^{(g)}\big) - \beta D_{\text{KL}}(\pi_\theta \,\|\, \pi_{\text{ref}})\Big], \tag{5}$$

where $r^{(g)}$ is the importance sampling ratio that measures how much the new model π_θ differs from the reference old model π_{ref} for the same sampled action $a^{(g)}$, defined as $r^{(g)} = \frac{\pi_\theta(a^{(g)}|s)}{\pi_{\text{ref}}(a^{(g)}|s)}$, and D_{KL} is the KL divergence between π_θ and π_{ref}, ensuring that updates do not push the model too far away from the reference behavior.

We partition all MFFCs into m mini-batches, and apply GRPO on each batch for n iterations[2], keeping the best restructuring solution before moving to the next batch. After all MFFCs are processed, the resulting restructured layout is passed back to the commercial flow for incremental legalization and further buffering and sizing.

5 EXPERIMENTAL RESULTS

5.1 Experimental Setup

Dataset and evaluation setup. We evaluate E-Layout on 9 diverse open-source designs from OpenCores [1], ITC'99 [16], Chipyard [5], and NVDLA [35]. All experiments are conducted using the ASAP 7 nm technology library within a leading commercial physical design flow. After E-Layout restructuring, the restructured layout is passed back to the commercial tool for incremental legalization and subsequent structure-invariant optimization, so the reported PPA reflects the effect of E-Layout after downstream placement optimization. We evaluate post-optimization TNS, WNS, and area. To ensure correctness, we employ formal verification at both the MFFC and full-chip levels: for MFFCs, we implement a combinational equivalence checker for standard-cell graphs using an SMT solver [20], and for full designs, we use Synopsys Formality for equivalence checking. During training and candidate evaluation, INSTA [30] is used to provide fast incremental layout TNS feedback.

[2]In practice, we set $m = 10 \sim 20$ and $n = 5 \sim 10$ to balance performance and runtime.

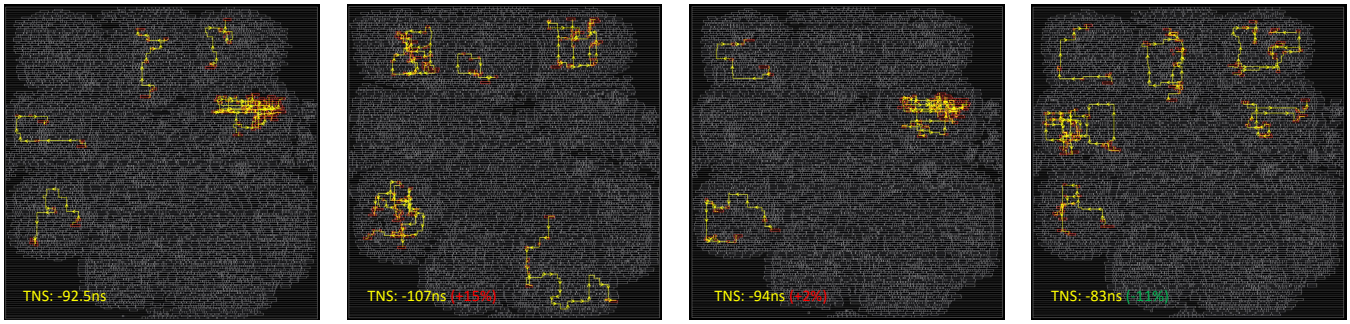
E-Layout implementation. E-Layout is implemented using `egg-log` [46] for e-graph equality saturation. For the graph RAG model, we use SGFormer [43] as the graph-transformer backbone for the retrieval encoder model, producing 768-dimensional embeddings, followed by MLP heads for sizing classification and location regression. The graph RAG model is further tuned with reinforcement learning using full-chip timing feedback. We adopt MFFCs as the basic restructuring granularity, and only optimize MFFCs with more than 5 equivalent logic candidates, thereby focusing optimization on cones with meaningful structural alternatives.

Baseline setup. We compare E-Layout against representative open-source layout restructuring flows, including OpenROAD [3] and OpenPhySyn [2]. To ensure fair comparison, all baselines start from the same post-placement design state and are followed by the same downstream commercial legalization and optimization flow as E-Layout. For OpenROAD [3], it identifies timing-critical regions and invokes ABC [7] for logic remapping, without physical-awareness. For OpenPhySyn, we enable its default restructuring-related transforms, including pin swapping and gate cloning.

5.2 Optimization Results

E-Layout restructuring results. Table 3 summarizes the post-placement PPA results across nine designs. E-Layout achieves on average 6.07% TNS improvement, 2.29% WNS improvement, and 1.07% area reduction over the commercial post-placement baseline. These gains are notable because the starting point has already undergone substantial commercial timing optimization, leaving limited room for further improvement.

Comparison with open-source layout restructuring tools. Compared with existing open-source restructuring flows, E-Layout delivers consistently stronger timing gains. OpenROAD-Restructure first identifies timing-critical regions after placement, but then delegates restructuring to ABC, whose logic remapping is performed without true layout awareness. As a result, the rewritten structures often break physical coherence with the surrounding layout and exhibit weak, or even negative, post-placement correlation, leading to average TNS degradation of up to 21.54%. OpenPhySyn-Restructure is more physically aware, but its optimization is limited to local cell-level edits such as pin swapping, gate cloning, buffering, and resizing. Because it does not explore meaningfully different logic structures, its improvement space remains narrow, resulting in 1.37% average TNS degradation and almost no WNS or area benefit. In contrast, E-Layout jointly optimizes equivalence-preserving



(a) Commercial baseline flow (b) w. OpenRoad-Restructure (c) w. OpenPhySyn-Restructure (d) w. E-Layout (ours)

Figure 5: Layout visualization of different methods. Top-20 critical paths are highlighted. Compared with the commercial baseline (a) and open-source restructuring flows (b)–(c), our E-Layout (d) delivers better timing results with fewer long detours.

Table 4: Statistics of E-Layout Restructuring.

Design-level		MFFC-level	
#. Cells	{423, 4320, 8257}	#. Restr. MFFCs	{17, 111, 200}
#. Restr. cells	{111, 263, 430}	- #. Expansion	{10, 80, 152}
- Percentage	{4%, 9%, 26%}	- #. Merge	{0, 3, 6}
#. Cells in MFFC	{1, 3, 27}	- #. Size only	{5, 28, 53}

These statistics are represented in format {min, avg, max}.

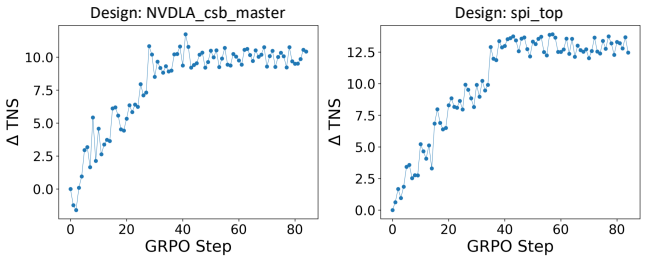


Figure 6: GRPO training curves demonstrating consistent TNS improvements over GRPO steps.

logic restructuring and physical realization under full-chip timing feedback, consistently improving both TNS and WNS while maintaining competitive area.

Some baseline entries are marked as unavailable because the corresponding open-source flows cannot support specific design constructs in our setup, such as multi-output gates. Nevertheless, the overall trend is clear: E-Layout provides the strongest and most consistent post-placement optimization among all compared methods.

5.3 Restructure Statistics

Table 4 summarizes the restructuring behavior of E-Layout. We only optimize MFFCs with more than 5 equivalent structures, among which about 9% are eventually rewritten. Each rewritten MFFC contains 3 cells on average. Although these cones are small, they are composed of complex standard cells rather than simple primitive gates (e.g., AIG), and can therefore admit rich combinational flexibility. In practice, even an MFFC with only a few gates can expand into many tens or even hundreds of equivalent alternatives after equality saturation. This makes such cones meaningful restructuring units in the post-placement setting: they are local enough to preserve physical coherence, yet still large enough to expose substantial structural diversity. Among the rewritten cones, roughly 70% are expanded from complex cells into compositions of simpler gates, 3% are merged into more complex cells, and the remaining 27% preserve the logic structure while only gate sizes are changed.

5.4 Case Study and Visualization

RL tuning trajectory. Figure 6 shows the GRPO training trajectories in terms of ΔTNS evaluated by INSTA. Every 5 GRPO steps, E-Layout optimizes a batch of MFFCs, selects the best restructuring solution, and then moves to the next batch. After one full pass over all candidate MFFCs, the second epoch converges quickly, indicating that the learned policy stabilizes and that the remaining timing improvement space becomes progressively smaller.

Design-level visualization. Figure 5 visualizes the top-20 critical paths for all methods after post-placement optimization. The commercial baseline and open-source restructuring flows still exhibit long, circuitous detours. Although OpenROAD performs aggressive remapping, its ABC-based optimization is not layout-aware and can even degrade timing. In contrast, E-Layout produces more direct critical paths and alleviates detours in dense regions, suggesting that the selected restructurings are better aligned with the surrounding physical context.

MFFC-level restructuring case studies. Figure 7 illustrates three representative cone-level decisions in E-Layout: expansion of a complex cell into simpler gates, merging of multiple gates into a more complex cell, and size-only tuning. These examples show that E-Layout does not favor a single restructuring pattern; instead, it adaptively chooses among structural expansion, merging, and physical-only adjustment according to local structure, physical context, and design-level timing feedback.

5.5 Ablation Study

We conduct ablation studies to evaluate the contribution of each component in the E-Layout framework, with results summarized in Figure 8. Removing the retrieval module and reducing the method to sizing-only optimization lowers the average TNS gain by about 3.5%, showing that selecting among equality-saturated logic alternatives is essential rather than merely resizing the original structure. Disabling the generation by using median sizes and random locations causes a further 2.4% drop, indicating the importance of jointly optimizing physical realization rather than treating it as a separate downstream step. Finally, removing RL fine-tuning and relying only on supervised tuning with local tool restructuring labels leads to the largest degradation, even worsening timing, which underscores the importance of full-chip timing feedback for aligning local restructuring decisions with the global objective.

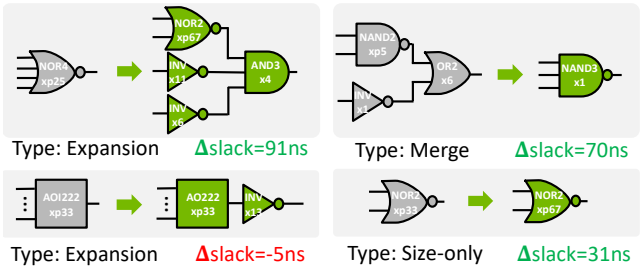


Figure 7: Case study of different MFFC restructuring types: expansion, merging, and sizing-only

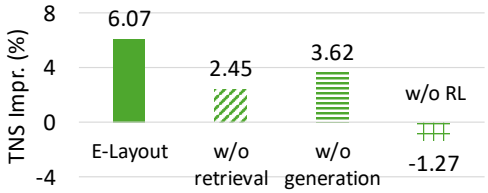


Figure 8: Ablation study of E-Layout ML framework on logic structure retrieval, physical attribute generation, and RL tuning for the graph RAG model.

5.6 Runtime and Scalability Analysis

Table 5 summarizes the runtime decomposition and scalability of E-Layout. The total runtime consists of two parts: local preprocessing and iterative optimization. The preprocessing stage includes netlist-to-graph conversion, MFFC extraction, and e-graph saturation. These steps are local to each MFFC, scale with design size and cone complexity, and are naturally parallelizable across CPU cores.

The iterative optimization stage includes graph RAG inference, RL tuning, and INSTA timing evaluation. Graph RAG inference is lightweight, completing in less than one second per batch on a single GPU, whereas RL tuning scales with the number of iterations and group size. In practice, incremental INSTA timing evaluation dominates the optimization loop. Overall, E-Layout follows a *local generation, global evaluation* workflow: local candidate generation scales well under parallel execution, while design-level timing evaluation is the main bottleneck, resulting in total runtime from minutes to hours depending on design size and optimization iterations.

6 DISCUSSION

Graph RAG as a learned extraction mechanism for e-graphs. Traditional e-graph optimization typically relies on rule-driven extraction with simple cost functions, such as area or delay, which are insufficient to capture the coupled logic–physical dependencies of post-layout timing. In E-Layout, the graph RAG model serves as a learned extraction mechanism: instead of selecting structures with a fixed linear cost, it retrieves promising logic alternatives and generates their physical realization conditioned on both logic and layout context. This allows e-graph extraction to be guided by richer logic–physical correlations than conventional heuristic cost functions.

Optimization granularity. The choice of MFFCs as the restructuring granularity reflects a practical tradeoff between optimization scope and physical controllability. If the region is too small, the available structural flexibility quickly collapses into sizing-like local adjustment; if it is too large, restructuring becomes increasingly entangled with neighboring logic and surrounding layout, making physically coherent implementation much harder. MFFCs occupy a useful middle ground: they are large enough to expose nontrivial logic alternatives, yet local enough to support incremental legalization and preserve the coherence of the surrounding placement. In principle, the same framework could be extended to finer or coarser regions, but doing so would require explicitly modeling stronger interaction across restructurings, especially for overlapping or mutually dependent optimization regions.

Toward unified layout optimization. At present, E-Layout focuses on ML-driven restructuring, while other layout-stage optimizations, such as buffering and sizing, are still handled by commercial tools. As a result, the final PPA can only be observed after the complete downstream flow, which makes closed-loop optimization relatively expensive. A natural next step is to move toward a unified framework in which restructuring, buffering, and sizing are optimized jointly under a shared timing objective. This could further reduce mismatch across stages and enable more direct optimization toward final layout quality.

Table 5: Runtime and scalability analysis of E-Layout.

Stage	Complexity / Scalability	Runtime	Parallel
Netlist-graph	Linear in number of gates	sec-min	
MFFC extract	Linear in MFFC gate number	sec-min	✓
E-graph rewrite	Scales with cone size/ equivalent space	sec-min	✓
RAG inference	Batched GPU inference	<1s/batch	✓
RL tuning	Scales with iterations and group size	sec-min	
INSTA eval.	Scales with design size	sec-min	✓
Overall	Scales with design size and iterations	min-hours	

7 CONCLUSION

We presented E-Layout, an ML-driven framework for post-placement layout restructuring. E-Layout preserves functional equivalence while jointly optimizing logic structure and physical realization under accurate layout timing. Its central contribution is to formulate layout-stage restructuring as a coupled logic–layout co-optimization problem spanning equivalence-preserving structure exploration, logic selection with physical realization, and design-level timing-driven refinement. By integrating e-graph equality saturation, graph RAG, and reinforcement learning with full-chip feedback, E-Layout enables structure-changing optimization directly at layout stage beyond conventional structure-invariant post-placement optimization. Evaluated on a 7 nm commercial physical design flow, E-Layout achieves on average 6% TNS improvement, 2% WNS improvement, and 1% area reduction, while consistently outperforming existing open-source layout restructuring tools.

8 ACKNOWLEDGEMENT

This work is supported by Hong Kong Research Grants Council (RGC) CRF-YCRG C6003-24Y, 16216825, and T46-415/25-R. It is also supported by RGC JLFS-YSF/E-605/26. It was partially conducted by ACCESS – AI Chip Center for Emerging Smart Systems, supported by the InnoHK initiative of the Innovation and Technology Commission of the Hong Kong Special Administrative Region Government.

REFERENCES

[1] [n. d.]. *OpenCores: The reference community for Free and Open Source gateware IP cores.*

[2] Ahmed Agiza and Sherief Reda. 2020. Openphysyn: An open-source physical synthesis optimization toolkit. In *2020 Workshop on Open-Source EDA Technology (WOSET).*

[3] Tutu Ajayi and David Blaauw. 2019. OpenROAD: Toward a self-driving, open-source digital layout implementation tool chain. In *Proceedings of Government Microcircuit Applications and Critical Technology Conference.*

[4] Charles J Alpert, Jiang Hu, Sachin S Sapatnekar, and CN Sze. 2006. Accurate estimation of global buffer delay within a floorplan. *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems* 25, 6 (2006), 1140–1145.

[5] Alon Amid, David Biancolin, Abraham Gonzalez, Daniel Grubb, Sagar Karandikar, Harrison Liew, Albert Magyar, Howard Mao, Albert Ou, Nathan Pemberton, et al. 2020. Chipyard: Integrated design, simulation, and implementation framework for custom SoCs. *IEEE Micro* (2020).

[6] Alan Mishchenko Robert Brayton and A Mishchenko. 2006. Scalable logic synthesis using a simple circuit structure. In *International Workshop on Logic and Synthesis (IWLS).*

[7] Robert Brayton and Alan Mishchenko. 2010. ABC: An academic industrial-strength verification tool. In *International Conference on Computer Aided Verification (CAV).*

[8] Cadence Design Systems. [n. d.]. Cadence Digital Full Flow Optimized to Deliver Improved Quality of Results and up to 3X Higher Productivity. https://www.cadence.com/en_US/home/company/newsroom/press-releases/pr/2020/cadence-digital-full-flow-optimized-to-deliver-improved-quality-.html.

[9] Shih-Chieh Chang, Kwang-Ting Cheng, Nam-Sung Woo, and Malgorzata Marek-Sadowska. 1997. Postlayout logic restructuring using alternative wires. *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems* 16, 6 (1997), 587–596.

[10] Chen Chen, Guangyu Hu, Cunxi Yu, Yuzhe Ma, and Hongce Zhang. 2025. Emorphic: Scalable Equality Saturation for Structural Exploration in Logic Synthesis. *arXiv preprint arXiv:2504.11574* (2025).

[11] Chen Chen, Guangyu Hu, Dongsheng Zuo, Cunxi Yu, Yuzhe Ma, and Hongce Zhang. 2024. E-syn: E-graph rewriting with technology-aware cost functions for logic synthesis. In *Design Automation Conference (DAC).*

[12] Chen Chen, Jiaqi Yin, and Cunxi Yu. 2025. Revisit Choice Network for Synthesis and Technology Mapping. *arXiv preprint arXiv:2508.14068* (2025).

[13] Chung-Kuan Cheng, Chester Holtz, Andrew B Kahng, Bill Lin, and Uday Mallappa. 2023. Dagsizer: A directed graph convolutional network approach to discrete gate sizing of vlsi graphs. *ACM Transactions on Design Automation of Electronic Systems* 28, 4 (2023), 1–31.

[14] Jianyi Cheng, Samuel Coward, Lorenzo Chelini, Rafael Barbalho, and Theo Drane. 2024. Seer: Super-optimization explorer for high-level synthesis using e-graph rewriting. In *Proceedings of the 29th ACM International Conference on Architectural Support for Programming Languages and Operating Systems, Volume 2.* 1029–1044.

[15] Chris Chu and Yiu-Chung Wong. 2007. FLUTE: Fast lookup table based rectilinear Steiner minimal tree algorithm for VLSI design. *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems* 27, 1 (2007), 70–83.

[16] Fulvio Corno, Matteo Sonza Reorda, and Giovanni Squillero. 2000. RT-level ITC'99 benchmarks and first ATPG results. *IEEE Design and Test of Computers* (2000).

[17] Samuel Coward, George A Constantinides, and Theo Drane. 2023. Automating constraint-aware datapath optimization using e-graphs. In *2023 60th ACM/IEEE Design Automation Conference (DAC).* IEEE, 1–6.

[18] Samuel Coward, Theo Drane, and George A Constantinides. 2024. Rover: Rtl optimization via verified e-graph rewriting. *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems* (2024).

[19] Tong Gao, Kuang-Chien Chen, Jason Cong, Yuzheng Ding, and C Liu. 1993. Placement and placement driven technology mapping for FPGA synthesis. In *Sixth Annual IEEE International ASIC Conference and Exhibit.* IEEE, 91–94.

[20] Marco Gario and Andrea Micheli. 2015. PySMT: a solver-agnostic library for fast prototyping of SMT-based algorithms. In *The International Workshop on Satisfiability Modulo Theories (SMT workshop).*

[21] Daya Guo, Dejian Yang, Haowei Zhang, Junxiao Song, Ruoyu Zhang, Runxin Xu, Qihao Zhu, Shirong Ma, Peiyi Wang, Xiao Bi, et al. 2025. Deepseek-r1: Incentivizing reasoning capability in llms via reinforcement learning. *arXiv preprint arXiv:2501.12948* (2025).

[22] Hao-Hsiang Hsiao, Yi-Chen Lu, Sung Kyu Lim, and Haoxing Ren. 2025. BUFFALO: PPA-Configurable, LLM-based Buffer Tree Generation via Group Relative Policy Optimization. In *Proceedings of the 44th IEEE/ACM International Conference on Computer-Aided Design (ICCAD).*

[23] Yi-Min Jiang, Angela Krstic, Kwang-Ting Cheng, and Malgorzata Marek-Sadowska. 1997. Post-layout logic restructuring for performance optimization. In *Proceedings of the 34th annual Design Automation Conference.* 662–665.

[24] Andrew B Kahng, Yiting Liu, and Zhiang Wang. 2025. Recursive Learning-Based Virtual Buffering for Analytical Global Placement. *arXiv preprint arXiv:2506.17247* (2025).

[25] Hosung Kim and John Lillis. 2008. A framework for layout-level logic restructuring. In *Proceedings of the 2008 international symposium on Physical design.* 87–94.

[26] Rongjian Liang, Siddhartha Nath, Anand Rajaram, Jiang Hu, and Haoxing Ren. 2023. Bufformer: A generative ml framework for scalable buffering. In *Proceedings of the 28th Asia and South Pacific Design Automation Conference.* 264–270.

[27] Junfeng Liu, Liwei Ni, Xingquan Li, Min Zhou, Lei Chen, Xing Li, Qinghua Zhao, and Shuai Ma. 2023. Aimap: Learning to improve technology mapping for asics via delay prediction. In *2023 IEEE 41st International Conference on Computer Design (ICCD).* IEEE, 344–347.

[28] Tianji Liu, Nutdranai Jaruthikorn, Shiju Lin, Bentian Jiang, Guannan Guo, Weihua Sheng, and Evangeline FY Young. 2025. Efficient and Effective E-graph-based Logic Optimization. *ACM Transactions on Design Automation of Electronic Systems* (2025).

[29] Yifang Liu, Rupesh S Shelar, and Jiang Hu. 2011. Simultaneous technology mapping and placement for delay minimization. *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems* 30, 3 (2011), 416–426.

[30] Yi-Chen Lu, Zhizheng Guo, Kishor Kunal, Rongjian Liang, and Haoxing Ren. 2025. Insta: An ultra-fast, differentiable, statistical static timing analysis engine for industrial physical design applications. In *2025 62nd ACM/IEEE Design Automation Conference (DAC).* IEEE, 1–7.

[31] Yi-Chen Lu, Kishor Kunal, Geraldo Pradipta, Rongjian Liang, Ravikishore Gandikota, and Haoxing Ren. 2025. LEGO-Size: LLM-Enhanced GPU-Optimized Signoff-Accurate Differentiable VLSI Gate Sizing in Advanced Nodes. In *Proceedings of the 2025 International Symposium on Physical Design.* 152–162.

[32] Yi-Chen Lu, Siddhartha Nath, Vishal Khandelwal, and Sung Kyu Lim. 2021. Rl-sizer: Vlsi gate sizing for timing optimization using deep reinforcement learning. In *2021 58th ACM/IEEE Design Automation Conference (DAC).* IEEE, 733–738.

[33] Siddhartha Nath, Geraldo Pradipta, Corey Hu, Tian Yang, Brucek Khailany, and Haoxing Ren. 2022. TransSizer: A Novel Transformer-Based Fast Gate Sizer. In *International Conference on Computer-Aided Design (ICCAD).*

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

[35] Nvidia. 2018. NVIDIA Deep Learning Accelerator. http://nvdla.org/primer.html

[36] Hongyang Pan, Cunqing Lan, Yiting Liu, Zhiang Wang, Li Shang, Xuan Zeng, Fan Yang, and Keren Zhu. 2024. Physically aware synthesis revisited: guiding technology mapping with primitive logic gate placement. In *Proceedings of the 43rd IEEE/ACM International Conference on Computer-Aided Design.* 1–9.

[37] Massoud Pedram and Narasimha B Bhat. 1991. Layout Driven Logic Restructuring/Decomposition.. In *ICCAD.* 134–137.

[38] Zhihong Shao, Peiyi Wang, Qihao Zhu, Runxin Xu, Junxiao Song, Xiao Bi, Haowei Zhang, Mingchuan Zhang, YK Li, Yang Wu, et al. 2024. Deepseekmath: Pushing the limits of mathematical reasoning in open language models. *arXiv preprint arXiv:2402.03300* (2024).

[39] Synopsys. [n. d.]. Fusion Compiler: RTL-to-GDSII Design Solution. https://www.synopsys.com/implementation-and-signoff/physical-implementation/fusion-compiler.html.

[40] Ecenur Ustun, Cunxi Yu, and Zhiru Zhang. 2023. Equality saturation for datapath synthesis: A pathway to Pareto optimality. In *2023 60th ACM/IEEE Design Automation Conference (DAC).* IEEE, 1–2.

[41] Lukas PPP Van Ginneken. 1990. Buffer placement in distributed RC-tree networks for minimal Elmore delay. In *1990 IEEE International Symposium on Circuits and Systems (ISCAS).* IEEE, 865–868.

[42] Hongxi Wu, Zhipeng Huang, Xingquan Li, and Wenxing Zhu. 2024. AiTO: Simultaneous gate sizing and buffer insertion for timing optimization with GNNs and RL. *Integration* 98 (2024), 102211.

[43] Qitian Wu, Wentao Zhao, Chenxiao Yang, Hengrui Zhang, Fan Nie, Haitian Jiang, Yatao Bian, and Junchi Yan. 2023. Sgformer: Simplifying and empowering transformers for large-graph representations. *Advances in Neural Information Processing Systems* 36 (2023), 64753–64773.

[44] Yuyang Ye, Peng Xu, Lizheng Ren, Tinghuan Chen, Hao Yan, Bei Yu, and Longxing Shi. 2024. Learning-driven physically-aware large-scale circuit gate sizing. *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems* (2024).

[45] Niansong Zhang, Chenhui Deng, Johannes Maximilian Kuehn, Chia-Tung Ho, Cunxi Yu, Zhiru Zhang, and Haoxing Ren. 2025. ASPEN: LLM-Guided E-Graph Rewriting for RTL Datapath Optimization. In *2025 ACM/IEEE 7th Symposium on Machine Learning for CAD (MLCAD).* IEEE, 1–9.

[46] Yihong Zhang, Yisu Remy Wang, Oliver Flatt, David Cao, Philip Zucker, Eli Rosenthal, Zachary Tatlock, and Max Willsey. 2023. Better together: Unifying datalog and equality saturation. *Proceedings of the ACM on Programming Languages* 7, PLDI (2023), 468–492.

[47] Quming Zhou and Kartik Mohanram. 2006. Gate sizing to radiation harden combinational logic. *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems* 25, 1 (2006), 155–166.