ResizerAgent: Agentic Strategy Selection for Timing Optimization in OpenROAD Resizer

Vidya A. Chhabria
Arizona State University
Tempe, AZ, USA
vachhabr@asu.edu

Atmadip Dey
Arizona State University
Tempe, AZ, USA
atmadipd@asu.edu

Andrew B. Kahng
UC San Diego
La Jolla, CA, USA
abk@ucsd.edu

Ioannis Savidis
Drexel University
Philadelphia, PA, USA
is338@drexel.edu

Pratik Shrestha
Drexel University
Philadelphia, PA, USA
ps937@drexel.edu

Bing-Yue Wu
Arizona State University
Tempe, AZ, USA
bingyuew@asu.edu

Abstract

Timing optimization in modern electronic design automation (EDA) tools requires strategies that effectively control heuristic engines and tool-specific input knobs. In OpenROAD, Resizer exposes timing repair operations and knobs, but selecting the right action depends on timing state, physical context, optimization history, and quality-of-results (QoR) metrics such as worst negative slack (WNS), total negative slack (TNS), area, power, and runtime. Prior autotuning methods search predefined knob spaces, but remain limited by the user-specified search space. We propose ResizerAgent, which uses large language model (LLM) reasoning to adaptively revise repair strategies, backtrack from regressions, and generate targeted fixes. An LLM-based planner-selector controller observes timing reports, design metrics, physical feedback, and prior outcomes to propose, evaluate, and refine repair strategies across iterations. By combining multi-plan exploration, physical awareness, and traceable reasoning, ResizerAgent enables strategy-level timing optimization. Experimental results show that ResizerAgent improves the effective clock period over an autotuned baseline with a comparable compute budget, while the planner-selector traces provide actionable insight into each repair decision that conventional autotuners do not offer.

CCS Concepts

• **Hardware → Timing analysis**; **Physical design (EDA)**; Software tools for EDA.

ACM Reference Format:
Vidya A. Chhabria, Atmadip Dey, Andrew B. Kahng, Ioannis Savidis, Pratik Shrestha, and Bing-Yue Wu. 2026. ResizerAgent: Agentic Strategy Selection for Timing Optimization in OpenROAD Resizer. In *2026 ACM/IEEE International Symposium on Machine Learning for CAD (MLCAD '26), September 07–09, 2026, Jeju Island, Republic of Korea.* ACM, New York, NY, USA, 9 pages. https://doi.org/10.1145/3831599.3840328

1 Introduction

Timing optimization is one of the most difficult and iterative tasks in digital implementation. Modern EDA tools rely on heuristic timing optimization engines whose effectiveness depends on the design, violation distribution, and optimization stage. These engines expose user-tunable knobs for repair effort, iteration count, path passes, repair budgets, endpoint selection, and the ordering of repair operations such as gate sizing, buffer insertion/removal, V_t swapping, pin swapping, and cell splitting. Thus, timing optimization requires not only an effective engine, but also an effective optimization strategy to invoke and set the values of these knobs.

We study this problem in OpenROAD [1], where timing optimization is primarily performed by Resizer via the `repair_timing` command [2]. Resizer implements several repair transformations and exposes knobs that control the scope and aggressiveness of timing repair. However, the default flow is necessarily general: it does not explicitly reason about the current timing state, previous repair attempts, or which strategy is most promising at a given iteration. Manual exploration of repair sequences and knob values is possible, but is expertise-intensive and computationally expensive.

Automatic tuning tools perform search over user-defined parameter spaces, e.g., AutoTuner [3] can explore values of selected knobs to improve a target objective. However, autotuning is limited by the search space specified before optimization begins. It does not naturally decide when to switch from broad timing repair to targeted path fixes, backtrack after a regression, or generate new repair scripts for the tool. Recent work has shown that LLM- based agents can enable reasoning-based optimization and outperform conventional autotuning [4].

In this work, we propose ResizerAgent, an LLM-based timing optimization flow for OpenROAD. The LLM does not replace Resizer; instead, the LLM acts as an adaptive controller around Resizer. ResizerAgent observes timing reports, quality-of-results (QoR) metrics, physical feedback, and previous optimization outcomes, then proposes the next repair strategy, generates the corresponding OpenROAD commands, and uses tool feedback to guide later iterations. The framework supports three strategies: custom sequences of Resizer repair operations with selected knob values; single-operation repair; and targeted fixes for specific violating paths or endpoints.

Our proposed flow follows a closed-loop agentic process. At each iteration, OpenROAD reports QoR metrics including worst negative slack (WNS), total negative slack (TNS), area, and power after legalization and updated global routing. A *planner* agent proposes candidate repair plans using these metrics and the optimization history, while a *selector* agent compares outcomes and chooses the next state. If the trajectory plateaus or all candidates regress, the selector can retry, change strategy, or backtrack to an earlier state. The planner-selector trace also improves interpretability by explaining why each plan is proposed, promoted, or rejected. Our contributions are as follows.

MLCAD '26, Jeju Island, Republic of Korea

ACM ISBN 979-8-4007-2878-5/2026/09
https://doi.org/10.1145/3831599.3840328

- We present ResizerAgent, which controls OpenROAD Resizer through a closed-loop agentic flow for timing optimization.
- We define three LLM-driven repair strategies: sequenced timing repair, single-operation repair, and targeted fixes.
- We introduce a planner-selector mechanism that uses timing metrics, physical feedback, and optimization history to choose repair actions, while generating reasoning traces that make timing optimization more interpretable than conventional optimization.
- We evaluate ResizerAgent against default Resizer, autotuned Resizer parameters, and ablated variants.
- At a similar compute budget, ResizerAgent outperforms AutoTuner in WNS and TNS, suggesting LLM reasoning is more effective than traditional parameter optimization for timing repair.

The public repository for ResizerAgent is at [5].

2 Background

2.1 Prior Work

Timing optimization in physical design has been studied for decades. Classical approaches use analytical delay models and mathematical optimization, including posynomial programming [6], convex delay modeling [7], Lagrangian relaxation (LR) [8], sensitivity-guided metaheuristics [9], and multithreaded LR-based discrete gate sizing [10]. More recent methods incorporate physical awareness through virtual buffering [11], differentiable placement-and-sizing [12], and simultaneous gate sizing and buffer insertion [13].

Machine-learning-based methods have also been proposed, including graph neural networks and imitation learning [14], transformers [15], reinforcement learning for gate sizing [16, 17], and generative-AI-based buffering [18]. While traditional and ML-based methods perform specific types of timing optimization, they often require tuning of algorithm hyperparameters, action spaces, or EDA tool knobs to achieve high quality of results (QoR).

Autotuning methods have emerged to automate this tuning process. These methods search over user-defined parameter spaces using algorithms such as reinforcement learning [19], recommender systems [20], Bayesian optimization, and grid search. In OpenROAD-flow-scripts, AutoTuner has been used to improve QoR by tuning flow parameters [3, 21]. However, autotuning approaches remain bounded by the parameter space specified before optimization begins. Autotuning approaches do not naturally decide when to change the optimization strategy, backtrack after a regression, or generate new repair scripts based on timing-report reasoning.

Advances in LLMs have enabled agentic AI approaches for EDA [4, 22, 23], in which LLM-based agents reason about tool outputs and iteratively refine chip design flows and chip design tools. Inspired by prior work that applied agentic AI to hyperparameter tuning of OpenROAD-flow-scripts [4], ResizerAgent tunes OpenROAD Resizer hyperparameters and performs targeted netlist modifications to improve QoR compared with conventional autotuning.

2.2 OpenROAD's Resizer

OpenROAD's Resizer [2] is a gate-level optimization engine used at multiple stages of the physical design flow to improve timing and electrical quality. It modifies the netlist using operations such as gate resizing, V_t swapping, buffer insertion and removal, gate cloning, load splitting, and input-pin swapping. In this work, we focus on the setup-repair mode of `repair_timing`.

The `repair_timing -setup` command identifies endpoints with negative setup slack and processes them in order of criticality. The `repair_tns` knob controls the percentage of timing endpoints to operate on (default value is 100). For each selected endpoint, Resizer analyzes the timing-critical path and attempts local repair operations in the order specified by the `sequence` knob (default sequence is {`unbuffer, vt_swap, sizeup, swap, buffer, clone, split`}). Repair effort is controlled by the `max_passes` and `max_iterations` knobs (default values are 10,000 passes and unlimited iterations). `max_passes` is a per-endpoint limit that bounds how many repair rounds Resizer may spend on the same selected endpoint. In one pass, Resizer examines the current critical path, attempts repair operations, updates timing after accepted changes, and checks whether the endpoint still violates setup timing. If the endpoint continues to violate, further passes may be performed up to the `max_passes` limit. By contrast, `max_iterations` is a global limit that bounds the total repair effort across all selected endpoints. Therefore, `max_passes` controls per-endpoint repair depth, while `max_iterations` controls the overall repair effort budget. The `setup_margin` knob defines the slack target at which an endpoint is considered repaired (default value is 0), and options such as `skip_last_gasp` and `skip_crit_vt_swap` disable late-stage aggressive repair (default value is false). Thus, Resizer is a heuristic timing-repair engine whose QoR depends on endpoint selection, operation ordering, and repair-effort limits. ResizerAgent uses these knobs as the interface for LLM-guided timing optimization.

3 ResizerAgent Architecture

We propose ResizerAgent, an LLM-driven timing optimization framework that drives OpenROAD's Resizer with adaptive optimization strategies and targeted netlist modifications. ResizerAgent starts from a design seed after clock tree synthesis (CTS) and iterates through four phases, as shown in Fig. 1. The *report phase* builds a structured prompt summarizing the current design state, prior optimization trajectory, and the impact of OpenROAD's default `repair_timing` command. The *plan phase* uses a planner agent to generate candidate optimization plans in a predefined JSON schema. The *execute phase* validates each plan, generates a TCL script for each plan, and runs it in OpenROAD. The *select phase* uses a selector agent to evaluate post-optimization metrics (WNS, TNS, power, and area), compare measured results against planner expectations, and choose the seed for the next iteration. The final solution is then passed through detailed routing to measure post-detailed-routing metrics. Implementation details are available in our GitHub repository [5].

3.1 Report Phase

The *report phase* begins each iteration from the chosen seed and gathers the design data from reports, logs, and the netlist. It combines them with markdown descriptions of the OpenROAD Resizer and the target PDK and the selector agent's feedback from the previous iteration. It constructs a prompt that is fed to the planner agent as context. This prompt is composed of the following sections. Example prompts are also available in [5].

(1) Design anchor and goal. This includes the design name, iteration number, clock-domain summary, and timing-closure target WNS ≥ 0. It also includes reference metrics from the default OpenROAD `repair_timing` command and from the current design.

(2) Worst path timing reports. We select the N_{ep} endpoints with the worst slack. For each endpoint, we collect up to $N_{\text{paths/ep}}$ worst paths, which form a *candidate list*. The list is then sorted by slack and the top N_{prompt} worst paths of the list are included in the prompt.

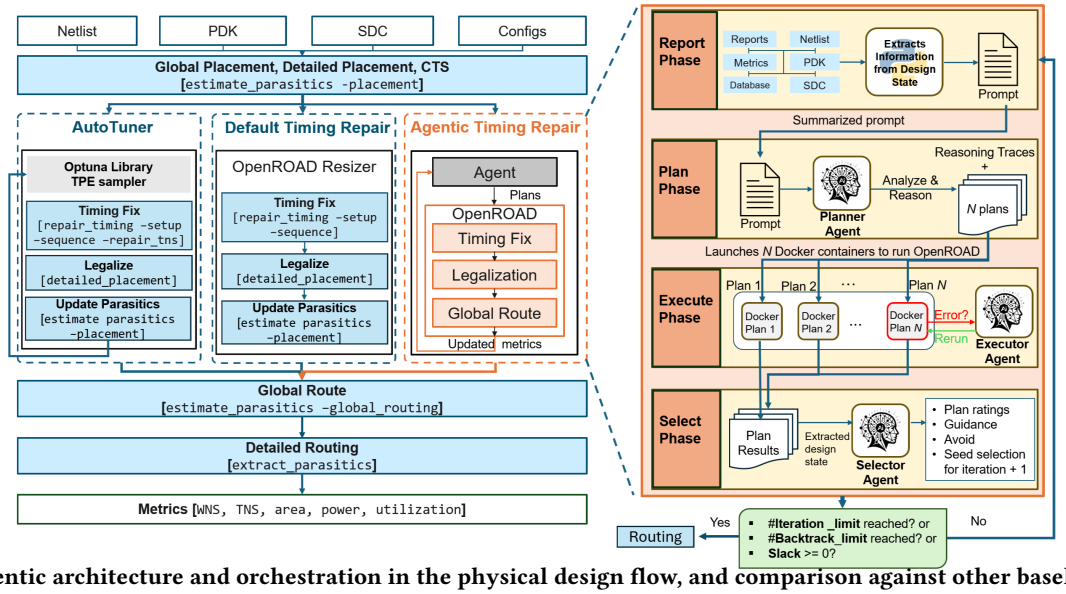

Figure 1: Agentic architecture and orchestration in the physical design flow, and comparison against other baseline engines.

(3) Endpoint slack histogram. The endpoints are sorted by slack and grouped into bins that each span B_{slack} ps. For each bin, we report the endpoint count and the number of unique startpoints. The histogram helps track slack distributions across iterations.
(4) Logical connectivity. For instances on the *candidate timing paths*, we extract one-hop logical connectivity from the netlist by collecting their immediate fanin and fanout instances. This includes neighboring instances that are not themselves on the candidate paths. This provides local connectivity context around critical paths.
(5) PDK constraints and Liberty cell catalog. We report the technology- and library-specific constraints that determine which repair operations are legal. For cells on the *candidate timing paths*, we provide a compressed cell catalog listing available drive strengths, threshold-voltage variants, and buffer options. This helps the planner avoid operations that are unavailable in the PDK or library.
(6) Physical information. *Placement context* includes die/core dimensions, instance count, total cell area, total site area, available site area, free placement sites in units of a 4×-sized buffer cell, and grid-based placement density (grid size $A_x \mu$m $\times A_y \mu$m). For candidate-path instances, we include centroid locations, local density, and candidate-path bounding boxes. From the second iteration onward, we also report average instance displacement across iterations using the displacement report in [13]. *Routing context* includes routed wirelength, via count, routed-net count, routing resource usage, overflow, top-congested layers, routing runtime, and routing layer range. Implementation details are available in [5].
(7) Previous iteration history. For iterations after the first, we summarize up to N_{hist} previous optimization attempts, including timing trajectory, attempted plans, metric changes, and selector ratings. We also compare the current worst endpoints with those from the previous iteration to identify endpoints that remain critical, have become critical, or have been resolved. Finally, we include selector feedback, strategies to avoid, and paths needing attention.

3.2 Plan Phase

The *plan phase* invokes the planner agent on the structured prompt generated by the report phase. The planner is stateless, so prior outcomes, current design context, and all information required for decision-making must be included in the prompt. The planner first reads the previous-iteration history to determine whether the trajectory is improving, stagnant, or regressing, and then reads the current seed's information as listed in Section 3.1.

The planner generates a JSON object following a predefined schema. The schema contains a decision field, a plan rationale, the number of generated plans, and a list of candidate plans. Each candidate specifies its type, repair actions, optional Resizer knobs, rationale, and expected impact on WNS, TNS, area, and power. The rationale and expected-outcome fields make the planner output auditable: after execution, the selector compares the predicted and measured WNS, TNS, area, and power to guide the next iteration. All plans are passed to the execute phase for validation and OpenROAD execution. The format supports three plan types:

- **Sequence plan.** Invokes OpenROAD's `repair_timing` command once with an ordered subset of timing operations and values for the Resizer knobs in Section 2.2.
- **Single-operation fix plan.** Applies up to $N_{\text{single-op}}$ consecutive `repair_timing` calls, each using one timing operation, with legalization and parasitic re-estimation between calls.
- **Targeted fix plan.** Applies up to N_{ECO} explicit ECO-style edits on the violating paths in the prompt to the planner, including resizing, V_t swapping, buffer deletion, and buffer insertion.

3.3 Execute Phase

The *execute phase* converts the planner agent's JSON output into runnable OpenROAD TCL scripts and evaluates each candidate plan. Each plan is translated using a plan-type-specific template. The *executor agent* uses the plan type, operation list, run knobs, and any target instances or nets specified by the planner. For a sequence plan, a single 'repair_timing' call is generated with the operation list and knob set. For a single-operation fix plan, one 'repair_timing' call is generated per operation. For a targeted fix plan, each operation is expanded into the OpenROAD primitive for that operation: `replace_cell` for `resize` timing operation, `remove_buffers` for `unbuffer` timing operation, and a custom buffer-insertion helper for `insert_buffer` timing operation. The generated scripts are evaluated independently by launching a Docker container per plan.

Every executed plan is followed by legalization, global routing, and metrics extraction using global-routing-based parasitics. The executor agent also handles OpenROAD TCL execution errors. Given the failed TCL script, planner JSON plan, and OpenROAD error log, it attempts to repair fixable script-level issues. If the repair fails, the plan is excluded from the selector agent's input.

3.4 Select Phase

The *select phase* closes the optimization loop by evaluating candidate-plan outcomes and providing guidance for the next iteration. The *selector agent* receives each plan's post-Opt metrics computed using global-routing-based parasitics, along with timing reports, the default OpenROAD-flow-scripts reference metrics, and recent iteration history. The selector is stateless, so all required context is included in its prompt.

Plan ranking. The selector uses a user-defined lexicographic policy to rank candidate plans. It ranks by WNS, treats plans within ϵ_{WNS} of the best WNS as equivalent, and breaks ties using TNS. If the TNS difference is within ϵ_{TNS}, it prefers the lower-power plan. This policy selects the best candidate plan, while the flow-control decision determines the design state for the next iteration.

Flow decisions and plateau handling. After ranking, the selector emits one of five flow decisions. (1) `continue` promotes the best-ranked plan as the next design state. (2) `retry` keeps the same design state and asks the planner to try a different strategy, typically when all candidates regress. (3) `accept-regression` promotes a regressing plan when a metric improves in a later iteration. (4) `backtrack` returns to an earlier promoted design state when the recent trajectory has plateaued. (5) `stop` terminates the loop when WNS ≥ 0 or two `backtrack` decisions have occurred. When emitting `backtrack`, the selector scans the iteration table in reverse to find the longest trailing window satisfying $|\Delta\text{WNS}| < \epsilon_{\text{WNS}}$ and $|\Delta\text{TNS}| < \epsilon_{\text{TNS}}$ at each iteration. The iteration immediately before this window becomes the backtrack target, excluding prior backtrack targets to prevent oscillation. The next iteration starts from that earlier design state, while the prompt preserves an *avoid list* of failed strategies from the abandoned tail.

The selector also diagnoses plateaus, using four labels. (1) `none` indicates continued progress through WNS/TNS improvement or movement of worst endpoints to less-critical slack bins. (2) `new-bottleneck` indicates that the worst endpoint has changed and should be targeted next. (3) `same-bottleneck` indicates that the same endpoint persists despite repairs. (4) `broad-stall` indicates that many endpoints remain in a slack band with no improvement.

Ratings and planner feedback. Ranking selects the best plan for the current iteration, while rating summarizes how each plan performed relative to the design state. The selector assigns each plan an *A/B/C/F rating* using thresholded changes in WNS, TNS, area, and power. An *A rating* requires WNS and TNS improvements of at least ϵ_{WNS} and ϵ_{TNS}, respectively, with area and power increase below ϵ_{area}% and ϵ_{power}%. A *B rating* indicates that either WNS or TNS improves over the threshold while the other remains within the threshold. A *C rating* indicates no meaningful change, and an *F rating* indicates WNS/TNS regression or execution failure.

The plan ratings are passed to the next planner invocation as feedback. The selector also emits a strategy note with actionable guidance, a stuck-path list, and an avoid list. The guidance identifies concrete operations, knob settings, paths, endpoints, or cells to target next. The stuck-path list records persistent endpoints and likely causes, such as limited drive-strength options or cells already at the lowest V_t tier. The avoid list records failed operations.

Table 1: ResizerAgent variables used in this work.

Variable	Definition	Value
N_{ep}	Number of violating endpoints selected by worst slack	50
$N_{\text{paths/ep}}$	Number of worst paths collected per selected endpoint	5
N_{prompt}	Number of worst-slack timing paths included in the prompt	25
B_{slack}	Slack-bin size for the endpoint slack histogram	20 ps
$A_x \times A_y$	Placement density grid-cell size to compute local density	40 μm × 40 μm
N_{hist}	Number of previous iterations' history in the prompt	3
N_{plans}	Maximum number of plans generated by planner	4
$N_{\text{single-op}}$	Maximum number of consecutive single-operation repair calls	9
N_{ECO}	Maximum number of explicit edits in a targeted fix plan	20
Π_{rank}	Lexicographic plan-ranking priority	WNS > TNS > power
ϵ_{WNS}	WNS equivalence and meaningful-change threshold	2 ps
ϵ_{TNS}	TNS meaningful-change threshold	5%
ϵ_{area}	Maximum allowable area increase for high-rated plans	5%
ϵ_{power}	Maximum allowable power increase for high-rated plans	5%

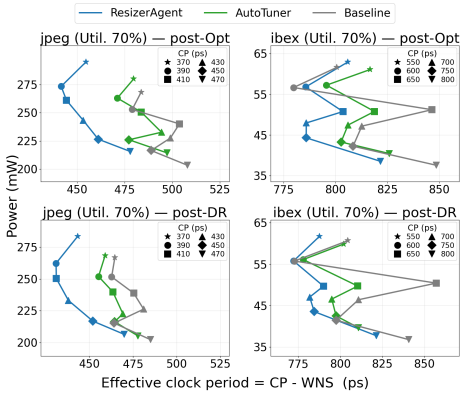

Figure 2: Power versus effective clock period curves for RA, AT, and BL flows across different target clock periods at fixed utilization on ASAP7 designs.

4 Experimental Setup

This section describes the setup used to evaluate ResizerAgent.

Tools. We use a modified version of OpenROAD [24] (based on commit hash: d1dde03 [25]) inside OpenROAD-flow-scripts (commit hash: 1ad9ee5 [21]).

Benchmarks. We use three benchmarks (aes, ibex, and jpeg) across two technologies, ASAP7 [26] and NanGate45 [21]. For each design, we generate multiple versions synthesized at different clock periods and utilization levels using OpenROAD-flow-scripts [21].[1]

ResizerAgent parameters. Table 1 summarizes the variables used to configure ResizerAgent. The variables include prompt-construction limits, planner output limits, selector thresholds, and the lexicographic ranking priority. These values are fixed across experiments unless otherwise stated. We use Claude Opus 4.7 for the *planner* and *selector agents* and Sonnet 4.6 for the *executor agent*. In our framework, these models can be interchanged with any other LLM.

Baselines. We evaluate ResizerAgent (RA) against two baselines.

(1) ORFS default baseline (BL). For the default baseline, we run the OpenROAD flow shown in Fig. 1 (blue boxes) using Resizer's default `repair_timing` parameter values in Section 2.2.

(2) AutoTuner baseline (AT). We use an AutoTuner setup that tunes the same parameters as those listed in Section 2.2. We use Optuna 3.x [27], an open-source hyperparameter-optimization library, configured with the Tree-structured Parzen Estimator (TPE) sampler [28]. The `sequence` parameter is a variable-length ordered list

[1] All experiments are performed on the following servers: (1) 512 GB RAM, AMD EPYC 7313 16-core processor, 32 physical cores, 64 logical cores; and (2) 995 GB RAM, AMD EPYC 7713 64-core processor, 64 physical cores, 128 logical cores.

Table 2: Comparison of RA, AT, and BL flows across target utilizations for ASAP7 designs. WNS and TNS are reported as positive magnitudes; smaller values are better.

Design CP (ps)	Util (%)	Stage	WNS (ps)			TNS (ns)			Area (μm^2)			Power (mW)			Tokens
			RA	AT	BL	RA	AT	BL	RA	AT	BL	RA	AT	BL	(Mils.)
aes 240 ps	60	post-Opt	48.98	49.98	49.89	5.41	5.50	5.70	2109	2092	2099	374.9	366.1	354.5	4.9
		post-DR	41.59	42.76	43.54	4.22	4.22	4.57	2109	2092	2099	364.9	356.1	359.2	-
	65	post-Opt	51.98	54.93	57.68	5.50	5.90	5.50	2097	2094	2104	372.8	368.2	357.2	4.7
		post-DR	45.30	46.86	50.83	4.48	4.80	4.35	2152	2094	2153	372.3	358.4	371.9	-
	70	post-Opt	49.27	52.21	55.19	5.10	5.40	5.30	2097	2071	2094	376.2	364.3	356.3	4.9
		post-DR	45.97	42.71	47.93	4.09	4.46	4.29	2097	2071	2094	368.2	355.4	362.8	-
ibex 750 ps	60	post-Opt	29.32	36.36	46.39	5.80	6.60	12.80	2470	2484	2456	43.5	44.3	40.4	6.6
		post-DR	24.39	32.79	45.22	0.56	0.38	2.92	2470	2484	2456	42.5	43.4	41.1	-
	70	post-Opt	41.72	45.64	60.31	8.40	13.70	15.00	2448	2451	2445	42.8	43.1	40.4	6.7
		post-DR	39.26	42.11	46.21	4.21	7.70	8.79	2448	2451	2445	42.0	42.3	41.2	-
	75	post-Opt	41.09	90.53	573.64	13.00	29.60	77.30	2431	2424	2430	42.1	40.6	39.2	6
		post-DR	38.73	65.38	97.43	5.64	14.39	24.97	2431	2424	2430	41.3	39.8	39.9	-
jpeg 430 ps	60	post-Opt	20.10	61.41	67.06	2.10	20.90	26.40	7072	7004	6955	247.4	241.3	231.2	5.3
		post-DR	8.25	35.33	49.26	0.08	6.39	11.97	7072	7004	6955	237.2	231.2	230.0	-
	70	post-Opt	22.64	47.48	69.00	2.60	14.20	25.00	7022	6957	6929	243.3	237.9	227.5	5.2
		post-DR	9.30	25.51	51.05	0.09	3.08	9.91	7022	6957	6929	233.1	227.8	226.3	-
	80	post-Opt	36.71	62.58	70.06	4.70	17.30	27.90	6938	6900	6924	242.3	238.8	229.4	5.2
		post-DR	26.09	48.80	58.37	1.07	6.80	15.46	6938	6900	6924	233.1	229.6	229.1	-

Table 3: Post-Opt and post-DR metrics for RA and the ablation baselines on ASAP7 designs. WNS and TNS are reported as positive magnitudes; lower is better.

Design	Stage	WNS (ps)				TNS (ns)			
		RA	A1	A2	A3	RA	A1	A2	A3
aes CP=160 ps Util.=65%	post-Opt	**124.1**	128.9	130.2	132.3	**17.85**	17.96	17.85	19.64
	post-DR	**122.8**	126.0	127.2	128.6	16.89	17.87	**16.79**	18.14
jpeg CP=330 ps Util.=70%	post-Opt	**124.4**	130.3	126.3	132.6	**64.61**	64.85	71.80	76.34
	post-DR	**110.0**	118.2	117.5	126.5	**55.06**	58.52	64.13	71.35
ibex CP=500 ps Util.=40%	post-Opt	**267.9**	278.8	276.1	274.2	330.00	312.57	**311.08**	321.53
	post-DR	**254.9**	265.0	270.1	268.20	299.49	309.23	**297.83**	323.42

Table 4: Per-iteration runtime comparison between RA and AT for ASAP7 designs under iso-compute budgets.

Design	CP (ps)	Util (%)	RA (s/iter)	AT Calib. (s)	AT (s/iter)
aes	240	70	412	2520	371
jpeg	430	80	1387	4798	1487
ibex	750	70	769	4301	871

drawn from the timing repair operations available, so it cannot be sampled as a primitive type. We encode sequence as 18 parameters: one binary include_move flag and one continuous weight_move $\in [1.0, 10.0]$ per operation, where each weight_move is sampled only when its corresponding include_move flag is 1. We decode the sequence by selecting the operations whose include_move flag is 1 and ordering them by descending weight; if all include_move flags happen to be zero, we force the operation with the highest weight to be on, so that every trial executes at least one repair action. The cost function is a weighted sum of WNS, TNS, and power (with positive magnitudes, and respective weights 0.4, 0.4, and 0.2), where the contribution of each metric M is $\frac{M_{\text{base}} - M_{\text{trial}}}{M_{\text{base}}}$.

Ablations. To justify the key architectural contributions of ResizerAgent, we evaluate three ablated variants.

Ablation 1 (A1): No targeted fixes. Disables the targeted fix plan, i.e., ResizerAgent can only modify the knobs in Section 2.2.

Ablation 2 (A2): Single candidate plan. Restricts ResizerAgent to generate only one candidate plan per iteration.

Ablation 3 (A3): No physical feedback. Removes physical information from the agent's inputs, including information from the post-global-routing evaluation feedback signal.

5 Evaluation of ResizerAgent

5.1 ResizerAgent vs. Baselines

We evaluate whether ResizerAgent's improvements generalize across different design operating points by sweeping both target clock period and target utilization. While we present the ASAP7 results in this section, the corresponding NanGate45 results are reported in Appendix A. In both experiments, we compare RA against AT and BL. Each flow is evaluated after timing repair, legalization, and global routing using global-routing-based parasitics (post-Opt), and again after detailed routing using extracted parasitics (post-DR). Results should be interpreted under RA's lexicographic objective, which prioritizes WNS and TNS over power. Since RA selects optimization plans based on post-global-routing timing estimates, detailed routing can reduce some of the observed post-Opt gains because global-routing-based and detailed-routing-based timing estimates are not perfectly correlated.

Across different target clock periods. For each design, we sweep the target clock period and evaluate the resulting power–timing tradeoff. Fig. 2 plots power versus effective clock period for ASAP7, where effective clock period is computed as the target clock period minus signed WNS. Fig. 2 (top row) shows that RA generally shifts the sampled power–timing tradeoff points toward lower effective clock periods compared with AT and BL. For both jpeg and ibex, RA achieves smaller effective clock periods than the two baselines across several target clock periods. RA uses higher power in some cases, particularly for jpeg, reflecting the lexicographic objective. The post-DR results in Fig. 2 (bottom row) preserve the same overall trend, although RA's gains over AT and BL are smaller.

Across different utilizations. Table 2 compares RA against AT and BL across target utilizations for ASAP7 designs. The table uses color coding to show QoR tradeoffs. For WNS and TNS, green marks the best value among the three flows. For area and power, colors indicate the cost of RA's timing improvement: green means RA improves or matches the metric, orange indicates modest overhead below 5%, and red indicates overhead above 5%. AutoTuner's objective also assigns greater weight to WNS and TNS than to power.

RA provides stronger timing recovery as utilization increases and timing closure becomes more physically constrained. RA shows its largest timing gains at several higher-utilization points, particularly for ibex and jpeg. For example, for ibex at 75% utilization,

Table 5: Representative *selector*-to-*planner* interactions illustrating recurring repair patterns on ASAP7.

Case / Pattern	Iter.	*Selector* Diagnosis	*Planner* Response	Result
Named-cell targeted fix aes; Util.: 70%, CP: 260 ps	2–3	Worst path shifted to endpoint `sa03_sr[5]` with three RVT cells.	Targeted fix V_t-swap or resize the exact named cells.	WNS: −34.39 → −28.26 ps TNS: −2.38 → −2.38 ns
Load-split pivot ibex; Util.: 75%, CP: 750 ps	10–11	Excessive load on the Han–Carlson adder chain.	Clone, buffer, sizeup, and V_t-swap focused on the chain.	WNS: −83.12 → −56.68 ps TNS: flat near −17.96 ns
Plateau break jpeg; Util.: 70%, CP: 430 ps	7–8	Same multiplier endpoint remained critical; over-buffered pre-logic chain was structurally exhausted.	Targeted fix on the heavily buffered launch chain feeding the stuck multiplier endpoint.	WNS: −43.86 → −26.62 ps TNS: −7.76 → −3.73 ns

RA improves both post-Opt WNS and TNS by more than 2× relative to AT. These results suggest that diagnosis-guided repair is particularly beneficial when higher utilization limits the effectiveness of generic timing repair and parameter tuning. The post-DR results preserve the same overall trend, although the post-Opt gains are reduced. The area and power columns show the corresponding tradeoff, with RA incurring additional physical cost to achieve these timing improvements.

Runtimes. Table 4 compares per-iteration runtimes under an iso-compute budget: RA runs up to 15 iterations with at most four plans each, while AT runs 13 iterations of four Optuna trial batches after a one-time calibration phase. We report single-iteration runtimes since total runtimes were collected on a shared machine. The RA value corresponds to a representative iteration with two sequence plans and one targeted fix plan, and is a slightly conservative estimate since targeted fix plans are faster. On a per-iteration basis, RA and AT are comparable; accounting for calibration, RA is faster overall.[2] At approximately the same compute budget, RA generally achieves better timing than AT, suggesting that LLM reasoning can be more effective than traditional hyperparameter search.

5.2 Ablation Studies of ResizerAgent

Table 3 reports post-Opt and post-DR WNS and TNS for RA and the three ablated variants. Across all designs and both stages, RA consistently achieves the lowest WNS, indicating that targeted fixes, multi-plan exploration, and physical feedback each contribute to WNS improvement. TNS, however, does not follow the same trend: A1 and A2 occasionally produce lower TNS than RA, particularly for ibex. For A1, this is explained by the absence of targeted fixes. Without targeted fixes, the agent applies only global Resizer knobs, which tend to reduce slack violations broadly across many paths rather than focusing repair effort on the single worst path. For A2, the single-candidate restriction similarly biases the agent toward plans with larger TNS impact. A3 consistently yields worse WNS and generally worse TNS than RA, confirming the importance of physical feedback.

5.3 Lexicographic Priority

ResizerAgent exposes the lexicographic ranking policy as a user-defined parameter along with the corresponding ϵ_{metric} values in Table 1. To demonstrate this flexibility, we evaluate a power-priority variant of RA where power is ranked above WNS and TNS in the selector's ranking policy; this contrasts with Table 2, where WNS is ranked first. Table 6 compares the two configurations. These results show that changing the ranking priority changes the selected power–timing tradeoff without modifying the framework.

[2]RA's per-iteration runtime may grow in later iterations as scope of timing improvement decreases, and the agents produce longer reasoning traces. Agents are bounded by model output speed (50–70 tokens/s for Claude Opus, 100+ for Claude Sonnet).

Table 6: Change in lexicographic priority on post-Opt metrics for ASAP7 designs. Power is prioritized above WNS and TNS.

Design	CP (ps)	Util (%)	RA WNS-priority			RA Power-priority		
			WNS (ps)	TNS (ns)	Power (mW)	WNS (ps)	TNS (ns)	Power (mW)
aes	240	70	49.27	5.10	376.2	47.52	5.61	370.0
jpeg	430	80	36.71	4.70	242.3	47.62	9.36	240.2
ibex	750	75	41.09	13.02	42.1	43.27	14.08	41.8

6 Discussion and Insights

Table 5 summarizes three representative *selector*-to-*planner* interactions as case studies. The examples illustrate how ResizerAgent uses the selector's diagnosis in determining the next repair action, rather than only searching over fixed knob values.

The first case, the **named-cell targeted fix** in aes, shows that ResizerAgent can localize a timing problem to specific cells. The selector identifies unswapped RVT cells on the new worst path, and the planner responds by applying V_t swaps or resizing to those cells. The second case, the **load-split pivot** in ibex, shows how the repair strategy changes when the selector identifies excessive load on the Han–Carlson adder chain. The planner then focuses cloning, buffering, sizing, and V_t swapping on the flagged chain. The third case, the **plateau break** in jpeg, shows how ResizerAgent responds when repeated repair no longer improves the same critical endpoint. The selector identifies an over-buffered pre-logic chain, and the planner moves to a targeted structural fix on the launch chain. These examples also show how the available standard-cell library affects the repair options. ASAP7 provides threshold-voltage variants, allowing the planner to use V_t swapping when appropriate. Corresponding behaviors on NanGate45, where multi-V_t cells are unavailable, along with additional interaction patterns, are discussed in Appendix A.3. Overall, the cases show that ResizerAgent goes beyond knob autotuning by using timing diagnosis to select targeted netlist modifications.

7 Conclusion

We have presented ResizerAgent, an LLM-based timing optimization framework that drives OpenROAD Resizer and performs targeted fixes through a closed-loop agentic flow. ResizerAgent observes timing reports, physical feedback, and optimization history to adaptively propose and refine repair strategies. Across designs, utilization levels, and target clock periods, it generally achieves better WNS and TNS than both the default OpenROAD flow and AutoTuner, while using approximately the same compute budget. The results show that RA can achieve better timing than conventional parameter search with a comparable compute budget. The ablation studies indicate that targeted fixes, multi-plan exploration, and physical feedback each contribute to the observed improvements.

References

[1] T. Ajayi, V. A. Chhabria, M. Fogaça, S. Hashemi, A. Hosny, A. B. Kahng, M. Kim, J. Lee, U. Mallappa, M. Neseem, G. Pradipta, S. Reda, M. Saligane, S. S. Sapatnekar, C. Sechen, M. Shalan, W. Swartz, L. Wang, Z. Wang, M. Woo, and B. Xu, "Invited: Toward an open-source digital flow: First learnings from the OpenROAD project," in *Proceedings of the ACM/IEEE Design Automation Conference*, 2019.

[2] "OpenROAD Resizer," https://github.com/The-OpenROAD-Project/OpenROAD/tree/master/src/rsz, 2026, accessed: 2026-08-14.

[3] J. Jung, A. B. Kahng, S. Kim, and R. Varadarajan, "METRICS2.1 and flow tuning in the IEEE CEDA robust design flow and OpenROAD ICCAD special session paper," in *Proceedings of the IEEE/ACM International Conference on Computer-Aided Design*, 2021.

[4] A. Ghose, A. B. Kahng, S. Kundu, and Z. Wang, "ORFS-agent: Tool-using agents for chip design optimization," in *Proceedings of the ACM/IEEE Symposium on Machine Learning for CAD*, 2025.

[5] "ResizerAgent-GitHub," https://github.com/ASU-VDA-Lab/ResizerAgent/tree/main, 2026, accessed: 2026-08-14.

[6] J. P. Fishburn and A. E. Dunlop, "TILOS: A posynomial programming approach to transistor sizing," in *Proceedings of the IEEE/ACM International Conference on Computer-Aided Design*, 1985.

[7] K. Kasamsetty, M. Ketkar, and S. Sapatnekar, "A new class of convex functions for delay modeling and its application to the transistor sizing problem [CMOS gates]," *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, vol. 19, no. 7, pp. 779–788, 2000.

[8] C.-P. Chen, C. Chu, and D. Wong, "Fast and exact simultaneous gate and wire sizing by lagrangian relaxation," *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, vol. 18, no. 7, pp. 1014–1025, 1999.

[9] J. Hu, A. B. Kahng, S. Kang, M.-C. Kim, and I. L. Markov, "Sensitivity-guided metaheuristics for accurate discrete gate sizing," in *Proceedings of the IEEE/ACM International Conference on Computer-Aided Design*, 2012.

[10] A. Sharma, D. Chinnery, T. Reimann, S. Bhardwaj, and C. Chu, "Fast lagrangian relaxation-based multithreaded gate sizing using simple timing calibrations," *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, vol. 39, no. 7, pp. 1456–1469, 2020.

[11] A. B. Kahng, Y. Liu, and Z. Wang, "Recursive learning-based virtual buffering for analytical global placement," in *Proceedings of the ACM/IEEE Symposium on Machine Learning for CAD*, 2025.

[12] Y. Du, Z. Guo, Y. Lin, R. Wang, and R. Huang, "Fusion of global placement and gate sizing with differentiable optimization," in *Proceedings of the IEEE/ACM International Conference on Computer-Aided Design*, 2024.

[13] A. B. Kahng, S. Kang, S. Kundu, Y. Liu, D. Markarian, S. Park, and Z. Wang, "Invited: Post-placement buffering and sizing contest," in *Proceedings of the International Symposium on Physical Design*, 2026.

[14] X. Zhou, J. Ye, C.-W. Pui, K. Shao, G. Zhang, B. Wang, J. Hao, G. Chen, and P. A. Heng, "Heterogeneous graph neural network-based imitation learning for gate sizing acceleration," in *Proceedings of the IEEE/ACM International Conference on Computer-Aided Design*, 2022.

[15] S. Nath, G. Pradipta, C. Hu, T. Yang, B. Khailany, and H. Ren, "TransSizer: A novel transformer-based fast gate sizer," in *Proceedings of the IEEE/ACM International Conference on Computer-Aided Design*, 2022.

[16] Y.-C. Lu, S. Nath, V. Khandelwal, and S. K. Lim, "RL-Sizer: VLSI gate sizing for timing optimization using deep reinforcement learning," in *Proceedings of the ACM/IEEE Design Automation Conference*, 2021, pp. 733–738.

[17] W. Jiang, V. A. Chhabria, and S. S. Sapatnekar, "IR-Aware ECO timing optimization using reinforcement learning," in *Proceedings of the ACM/IEEE Symposium on Machine Learning for CAD*, 2024.

[18] R. Liang, S. Nath, A. Rajaram, J. Hu, and H. Ren, "BufFormer: A generative ML framework for scalable buffering," in *Proceedings of the Asia and South Pacific Design Automation Conference*, 2023.

[19] A. Agnesina, K. Chang, and S. K. Lim, "VLSI placement parameter optimization using deep reinforcement learning," in *Proceedings of the IEEE/ACM International Conference on Computer-Aided Design*, 2020.

[20] J. Kwon, M. M. Ziegler, and L. P. Carloni, "A learning-based recommender system for autotuning design flows of industrial high-performance processors," in *Proceedings of the ACM/IEEE Design Automation Conference*, 2019.

[21] "OpenROAD flow scripts," https://github.com/The-OpenROAD-Project/OpenROAD-flow-scripts, 2026, accessed: 2026-08-14.

[22] B.-Y. Wu, A. Dey, A. Rovinski, and V. A. Chhabria, "Focus session: Large language models in physical design: From data generation to intelligent agents," in *Proceedings of the Design, Automation & Test in Europe*, 2026.

[23] A. Ghose, A. B. Kahng, S. Kundu, and B. Pramanik, "Invited: Agentic AI for physical design R&D: Status and prospects," in *Proceedings of the International Symposium on Physical Design*, 2026.

[24] "Modified ORFS," https://github.com/Atmadip/ORFS, 2026, accessed: 2026-08-14.

[25] "OpenROAD," https://github.com/The-OpenROAD-Project/OpenROAD, 2026, accessed: 2026-08-14.

[26] L. T. Clark, V. Vashishtha, L. Shifren, A. Gujja, S. Sinha, B. Cline, C. Ramamurthy, and G. Yeric, "ASAP7: A 7-nm FinFET predictive process design kit," *Microelectronics Journal*, vol. 53, pp. 105–115, 2016. [Online]. Available: https://www.sciencedirect.com/science/article/pii/S002626921630026X

[27] T. Akiba, S. Sano, T. Yanase, T. Ohta, and M. Koyama, "Optuna: A next-generation hyperparameter optimization framework," in *Proceedings of the ACM SIGKDD International Conference on Knowledge Discovery & Data Mining*, 2019.

[28] J. Bergstra, R. Bardenet, Y. Bengio, and B. Kégl, "Algorithms for hyper-parameter optimization," in *Advances in Neural Information Processing Systems*, 2011.

[29] "ResizerAgent-Zenodo," https://zenodo.org/records/21959752, 2026, accessed: 2026-08-14.

A Appendix for Evaluation on NanGate45

NanGate45 is a single-V_t library with fewer gate-size variants than ASAP7, which restricts the available repair operations: threshold-voltage swapping is not possible. The agent relies entirely on gate sizing, buffer insertion, and structural fixes. Despite this reduced optimization scope, the broad trends observed on ASAP7 carry over to NanGate45, as reported below.

A.1 Across Different Target Clock Periods

Fig. 3 plots power versus effective clock period (ECP) for RA, AT, and BL across a sweep of target clock periods at fixed utilization on the ibex NanGate45 design (same format as Fig. 2). The top row reports post-Opt results and the bottom row reports post-DR results. For ibex, the three flows produce overlapping power–ECP curves, making the global separation less visible than on ASAP7. However, when comparing the three flows at the same target clock period, RA achieves a smaller effective clock period than AT and BL in most cases, indicating better timing recovery at a small power overhead. This tradeoff is consistent with RA's lexicographic objective, which prioritizes WNS before power.

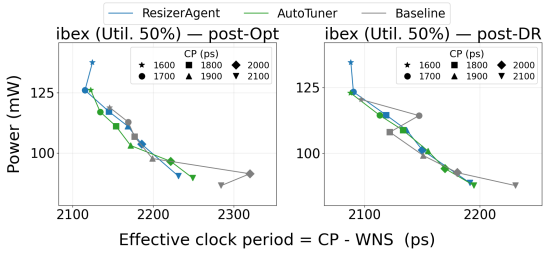

Figure 3: Power versus effective clock period curves for RA, AT, and BL flows across different target clock periods at fixed utilization on NanGate45 ibex.

A.2 Across Different Utilizations

Table 7 reports the full utilization sweep for NanGate45 designs, following the same experimental setup and color coding as Table 2. RA achieves the lower WNS across most designs and utilization levels, with the largest gains at higher utilization. Since NanGate45 is a single-V_t library, threshold-voltage swapping is unavailable, so RA relies more heavily on gate sizing, buffer insertion, and targeted structural fixes.

A.3 Discussion on NanGate45 Designs

Table 8 shows some repair patterns specific to NanGate45, complementing the ASAP7 interactions discussed in Section 6. The first pattern concerns negative feedback. The aes case shows that when broad plans increased TNS, the agent narrowed the action space to a minimal `sizeup`/`sizeup_match` sequence. The ibex case shows harmful-operation avoidance, where the agent avoided a behavior that improved WNS but worsened TNS. The jpeg case shows over-buffering correction, where the agent bypassed buffer and clone operations and instead targeted small `INV_X1` drivers directly. Across all three cases, negative feedback from the *selector* is as important as positive feedback: the *planner* suppresses behaviors that degrade the timing distribution.

The second pattern concerns library awareness. Unlike ASAP7, which provides threshold-voltage variants that allow the *planner* to use `vt_swap` when the *selector* identifies unswapped RVT cells, NanGate45 is a single-V_t library where threshold-voltage swaps are unavailable. As a result, the agent relies entirely on sizing, buffering, and structural fixes. Despite the reduced optimization scope, ResizerAgent still achieves meaningful WNS improvement across all three NanGate45 cases.

B Artifact Appendix

B.1 Abstract

This appendix describes the artifact released with this paper and archived on Zenodo [29]. The archived artifact contains the source code, experiment scripts, datasets, configurations, prompts, and cached outputs used for the artifact evaluation (AE) at MLCAD 2026. The latest version of ResizerAgent is available on GitHub [5].

B.2 Artifact Checklist (Meta-Information)

- **Algorithm:** Agentic physical design optimization using tool calling, hyperparameter tuning, and targeted netlist modifications.
- **Models:** Anthropic Claude Opus 4.7 and Sonnet 4.6.
- **Model availability and access:** Closed models accessed through Claude Code; a Claude Code subscription is required.
- **LLM prompts and inference settings:** Prompts and sample agent outputs are provided in the archived artifact [29].
- **Dataset:** Three designs (aes, ibex, and jpeg) evaluated across multiple target clock periods and utilizations using ASAP7 and NanGate45.
- **Runtime environment:** Docker.
- **Container, VM, or locked environment:** Docker container.
- **Minimum hardware configuration:** 16-core CPU, 64 GB RAM, and 10 GB disk space.
- **Execution:** Experiment scripts and instructions are provided in the archived artifact [29].
- **Metrics:** WNS, TNS, area, and power.
- **Output:** Agent outputs in JSON format and OpenROAD outputs include Verilog, DEF, and ODB files.
- **Experiments:** Clock-period sweep, utilization sweep, ablation studies, and lexicographic-priority experiments across both technology platforms.
- **Disk space required (approximately):** 2–3 GB per design configuration.
- **Time needed to prepare workflow (approximately):** Docker installation and artifact setup.
- **Time needed to complete experiments (approximately):** 2–3 hours for a single ResizerAgent run on the largest design and approximately 5–6 days for the complete experiment set when run serially.
- **API cost or GPU-hours required (if applicable):** Approximately $50 of Claude Code usage for a single ResizerAgent run on a single design.
- **Publicly available:** Yes [5, 29].
- **Code license:** BSD 3-Clause License.
- **Workflow framework used:** Claude Code.
- **Zenodo DOI for archived artifact:** 10.5281/zenodo.21959752.

B.3 Description

- **How to access:** The artifact associated with this paper is archived on Zenodo [29]. The latest version of ResizerAgent is on GitHub [5].
- **Hardware dependencies:** The minimum hardware requirements are a system with a 16-core CPU and 64 GB RAM. Approximately 2–3 GB of disk space is required per design configuration.
- **Software dependencies:** The experiment environment is provided through Docker. Installation instructions and required dependencies are documented in the archived artifact [29].
- **Commercial software and PDK dependencies:** None.
- **Datasets:** All design inputs and experiment configurations used in the paper are included in the archived artifact [29].
- **Models:** The planner and selector use Claude Opus 4.7, while the executor uses Sonnet 4.6. Model calls were performed through Claude Code using a Claude Code Max subscription. The reported experiments were conducted from May 10 to May 20, 2026. Prompts, cached outputs, and experiment logs are also in the artifact [29].

Table 7: Comparison of RA, AT, and BL flows across different target utilizations for NanGate45 designs, complementing Table 2. WNS and TNS are reported as positive magnitudes; smaller values are better.

Design CP (ps)	Util (%)	Stage	WNS (ps)			TNS (ns)			Area (μm^2)			Power (mW)			Tokens
			RA	AT	BL	RA	AT	BL	RA	AT	BL	RA	AT	BL	(Mils.)
aes 620 ps	40	post-Opt	155.10	154.20	211.90	18.77	20.48	19.67	26301	24617	25537	562.20	512.10	519.50	5.1
		post-DR	135.60	147.20	140.80	16.98	19.06	17.75	26301	24617	25537	552.85	502.93	530.41	-
	45	post-Opt	153.00	164.40	235.30	18.99	20.39	21.28	25181	25167	24683	527.80	529.00	496.00	4.7
		post-DR	143.70	161.30	159.70	17.29	19.19	19.99	25181	25167	24683	519.30	520.45	507.33	-
	50	post-Opt	175.80	196.20	254.80	19.55	21.83	21.53	26394	24491	25376	571.70	503.80	513.40	4.6
		post-DR	176.80	192.80	170.90	17.69	19.94	19.22	26394	24491	25376	563.45	495.63	525.95	-
ibex 2100 ps	50	post-Opt	121.60	149.20	184.60	10.25	21.29	16.83	30762	30434	30402	93.03	89.58	86.54	5.6
		post-DR	86.50	94.90	130.80	3.36	16.63	8.74	30762	30434	30402	91.20	87.71	87.72	-
	60	post-Opt	115.60	155.10	175.90	28.60	24.39	34.03	30480	30263	30077	90.62	87.85	83.26	5.7
		post-DR	79.20	83.90	110.60	11.24	13.87	21.14	30480	30263	30077	88.66	85.94	84.75	-
	70	post-Opt	125.80	167.70	155.80	12.22	21.46	22.92	30405	30218	30105	89.41	87.89	83.18	5.5
		post-DR	78.60	85.20	104.30	2.72	10.64	8.44	30405	30218	30105	87.57	85.82	84.74	-
jpeg 1000 ps	50	post-Opt	159.70	156.30	162.60	70.83	70.96	73.79	95081	94829	94842	591.80	587.20	566.00	4.87
		post-DR	131.90	130.30	130.50	55.57	56.30	57.94	95081	94829	94842	568.14	563.45	563.80	-
	60	post-Opt	135.90	141.90	141.80	64.71	63.79	63.95	95322	94544	94540	593.40	584.60	562.30	4.79
		post-DR	117.20	120.10	118.60	51.77	48.84	48.17	95322	94544	94540	569.37	560.96	560.83	-
	70	post-Opt	164.50	152.50	151.00	76.45	66.83	68.92	94405	94909	94471	585.30	593.70	564.30	5.08
		post-DR	144.80	125.60	132.90	64.73	55.00	56.65	94405	94909	94471	563.20	571.65	563.93	-

Table 8: Representative *selector*-to-*planner* interactions illustrating recurring repair patterns on NanGate45, complementing Table 5. The cases highlight patterns specific to a single-V_t library, where threshold-voltage swapping is unavailable.

Case / Pattern	Iter.	*Selector* Diagnosis	*Planner* Response	Result
Minimal repair after regression aes; Util.: 50%, CP: 620 ps	3–4	Broad structural sequences were increasing TNS.	Planned targeted fix and `sizeup`/`sizeup_match` sequence.	WNS: −254.3 → −178.9 ps TNS: −22.49 → −19.44 ns
Harmful operation avoidance ibex; Util.: 70%, CP: 2100 ps	1–2	Previous plan improved WNS but worsened TNS after late-stage buffer insertion created additional violating paths.	Avoided late-stage buffer insertion; used `sizeup_match` and plans that start with `unbuffer`.	WNS: −176.2 → −155.6 ps TNS: −59.27 → −25.30 ns
Over-buffering correction jpeg; Util.: 70%, CP: 1000 ps	3–4	Over-buffering detected: five `BUF_X32` stages in sequence.	Avoided buffer and clone; targeted specific small `INV_X1` drivers.	WNS: −214.0 → −164.5 ps TNS: −103.75 → −76.45 ns

B.4 Installation

Instructions for installing the required environment, configuring Claude Code, running ResizerAgent, and generating the experiment results are provided in the README files in the artifact [29].

B.5 Experiment Workflow

The experiment workflow consists of four stages. First, the `base` stage runs the physical design flow through CTS and generates the starting design state. Second, the `default` stage runs the baseline timing optimization. Third, the `llm_iterations` stage runs ResizerAgent, which iteratively analyzes the design, generates repair plans, executes OpenROAD commands, and selects the best solution. Finally, the `backend` stage completes detailed routing, to evaluate the selected solution post-DR. Scripts and instructions for executing each stage are provided in the archived artifact [29].

B.6 Evaluation and Expected Results

Running `run.py` generates the post-Opt artifacts and experiment results under `work_dir/<agent>/<pdk>/<design>/`. The `base/`, `default/`, and `llm_iterations/` directories contain the design states generated during the corresponding stages of the workflow. The generated outputs include ODB and DEF files, as well as CSV files containing the selected `repair_timing` hyperparameters and the resulting WNS, TNS, power, and area. The ranking of candidate repair plans and the selected solution are in `best_solutions/rankings.json`. Experiment inputs are provided under `AE/dataset/<pdk>/<design>/<config>/` and are described in `AE/dataset/README.md`. The `outputs/` directory provides cached results; correspondence to the tables and figures in the paper is documented in `outputs/README.md`.

B.7 Experiment Customization

Users can modify the provided config files to evaluate ResizerAgent for different designs. Instructions for configuring and running these experiments are provided in [29].

B.8 Methodology

Submission, reviewing, and badging methodology used for AE:

- https://www.acm.org/publications/policies/artifact-review-and-badging-current
- http://cTuning.org/ae/submission-20201122.html
- https://github.com/ml-eda/artifact-evaluation/