TAPCO: LLM-Guided Design-Adaptive Timing and Power Co-Optimization

Wuqian Tang*
National Tsing Hua University

Zixiao Wang*†
The Chinese University of Hong Kong

Che-Chien Lin
National Tsing Hua University

Fong-Hua Fu
National Tsing Hua University

Laura Angélica Shion-Korsak
National Tsing Hua University

Xinyun Zhang
ShanghaiTech University

Bei Yu
The Chinese University of Hong Kong

Chun-Yao Wang†
National Tsing Hua University

Abstract

Post-placement timing–power optimization requires navigating a design-dependent space of repair actions under tightly coupled electrical and physical constraints. Existing rule-based optimizers rely on fixed repair policies and exposed controls, making complex design priors difficult to express, performance sensitive to hyperparameters, and design-specific corner cases dependent on expert intervention. We present TAPCO, an LLM-guided framework for autonomous optimizer adaptation. TAPCO uses the LLM's implicit prior over physical-design code and optimization strategies to identify promising directions, while an explicit feedback-driven process encoded in the prompt grounds each proposal in measured outcomes. Its hierarchical search first adapts the existing configuration and then, when necessary, the optimizer policy itself, expanding the effective search space beyond conventional hyperparameter tuning. Evaluation on an independently assessed eight-design ASAP7 suite demonstrates substantial timing and power improvements and reveals distinct design-specific optimization strategies. These results show that LLM priors, when coupled with objective evaluator feedback, can automate optimizer adaptations that otherwise require manual engineering effort.

CCS Concepts

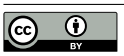
• **Hardware → Electronic design automation**; • **Computing methodologies → Artificial intelligence**.

Keywords

MLCAD 2026 Contest, large language models, physical design, timing-power co-optimization

ACM Reference Format:
Wuqian Tang, Zixiao Wang, Che-Chien Lin, Fong-Hua Fu, Laura Angélica Shion-Korsak, Xinyun Zhang, Bei Yu, and Chun-Yao Wang. 2026. TAPCO: LLM-Guided Design-Adaptive Timing and Power Co-Optimization. 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, 8 pages. https://doi.org/10.1145/3831599.3840364

*Equal contribution.
†Corresponding authors.



This work is licensed under a Creative Commons Attribution 4.0 International License.
MLCAD '26, Jeju Island, Republic of Korea
© 2026 Copyright held by the owner/author(s).
ACM ISBN 979-8-4007-2878-5/2026/09
https://doi.org/10.1145/3831599.3840364

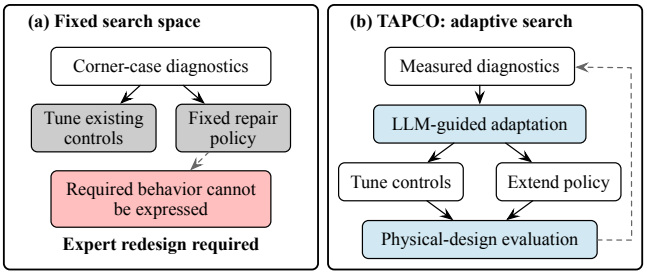


Figure 1: Fixed versus adaptive optimizer search. Conventional tuning remains within predefined controls and repair logic; corner cases outside this space require expert redesign. TAPCO uses measured feedback to tune controls and, when needed, extend policy under physical-design evaluation.

1 Introduction

Post-placement timing and power optimization must improve a design without exhausting the limited physical headroom left after placement. Cell sizing and buffering can recover slack while increasing power, placement demand, and congestion, whereas low-threshold-voltage cells improve delay at the cost of leakage [2, 3, 6, 21]. The value of each action depends on the current slack distribution, density, cell mix, and whitespace. Consequently, successful optimization requires adapting not only individual repair actions but also how they are scheduled and constrained for the design at hand.

Modern physical-design tools provide a rich collection of rule-based repair operators and encode considerable engineering knowledge in hand-crafted schedules and exposed control parameters [1, 22]. Operators define the available physical transformations, while the optimization policy decides when and where to apply them, how to allocate repair effort, and which intermediate improvements to retain. Existing rule-based flows have three limitations at this policy layer. First, fixed interfaces cannot easily encode complex priors about design state or observed failure patterns. Second, interacting margins, budgets, move switches, and utilization limits make performance sensitive to design-dependent hyperparameters. Third, conventional tuning searches a space fixed before optimization begins and therefore assumes that the required behavior is

already expressible by existing controls. This assumption often fails on dense or irregular corner cases, where a flow can repeatedly make low-value moves or reject useful partial progress. Engineers escape such states by revising the schedule, effort allocation, or commit logic—thereby changing the effective search space rather than merely selecting new parameter values.

Large language model (LLM) agents provide a promising mechanism for this role. Their pretrained priors over code, program behavior, and optimization strategies allow them to connect source, diagnostics, and measured outcomes when reasoning about a stalled optimization process [7, 18, 23, 24]. An LLM can therefore hypothesize why a policy fails and propose a targeted parameter or program change. Such open-ended proposals must nevertheless remain grounded in physical-design evidence: the LLM identifies a promising direction, while the domain evaluator determines whether the candidate is feasible and beneficial.

We introduce TAPCO, an LLM-guided framework that adapts post-placement timing–power optimizers to individual designs. As shown in Fig. 1, TAPCO treats the optimizer itself as the object of iterative adaptation. It first searches exposed controls and, when they cannot express the needed behavior, extends adaptation to the policy that schedules repair effort and retains intermediate outcomes. An explicit feedback-driven process encoded in the prompt organizes this search: each iteration interprets measured results, proposes a bounded configuration or policy change, evaluates the candidate on the physical design, and conditions the next proposal on the outcome. The evaluator, rather than the LLM's confidence, determines progress and feasibility. TAPCO can thus specialize both parameters and policies to each design while expanding the search space only when supported by observed evidence.

We evaluate TAPCO on an independently assessed suite of eight ASAP7 designs. TAPCO delivers substantial timing and power improvements, including on designs for which parameter adaptation alone is insufficient. The resulting optimizers exhibit distinct design-specific strategies at both configuration and policy levels.

The main contributions of this work are as follows.

- We formulate design-specific adaptation as joint search over configurations and repair policies beyond fixed parameter spaces.
- We develop TAPCO, which combines an LLM program prior with evaluator-grounded sequential feedback and hierarchical policy adaptation.
- We demonstrate quality-of-result gains on eight ASAP7 designs and analyze adaptation beyond hyperparameter tuning.

2 Preliminaries

We review TAPCO's physical-design and search foundations, then formulate its design-specific adaptation objective.

2.1 Timing–Power Co-Optimization

Post-placement optimization acts on a physically instantiated design whose timing, power, and legalization constraints are already tightly coupled. Common transformations include gate sizing and VT substitution, buffering, cloning, pin swapping, and legalization [2, 6, 9, 17, 21]. Their benefits depend on both electrical effect and physical state: upsizing, low-VT substitution, and buffering can recover timing but consume power, capacitance, whitespace, or routing resources [20, 25]. The repair policy must therefore choose endpoints, order transformations, allocate effort, and decide which partial improvements to retain according to the current design state.

Evaluation couples timing recovery and dynamic- and leakage-power reduction with electrical, displacement, routing, and runtime costs. Placement legality and logic equivalence instead define hard feasibility: an electrically faster candidate may still be unusable. This distinction follows the benchmark evaluator used here [4].

Rule-based tools encode substantial expertise in both their physical operators and their repair policies [1, 22]. Conventional autotuning improves such a flow by searching exposed margins, budgets, move switches, and utilization limits [10, 26]. This configuration search is effective when the fixed policy already expresses the needed behavior. TAPCO additionally adapts policy decisions such as endpoint scheduling, effort allocation, and partial-gain retention when it does not.

2.2 Black-Box Search and Program Priors

Optimizer adaptation presents an expensive black-box search problem. A configuration change requires complete physical evaluation; a program change must also be built and can introduce functional, legality, or runtime failures. Bayesian optimization addresses such expensive settings by conditioning a surrogate on measured observations and balancing exploration against exploitation [14, 19]. Its multi-objective variants have been applied to physical-design parameter tuning [10, 26]. More generally, this sequential principle maintains an incumbent and uses every measured outcome, including failures, to select the next experiment.

Program adaptation introduces a complementary requirement: the proposal mechanism must reason about structured code rather than only numeric samples. Coding agents provide an implicit prior over program behavior and debugging strategies and condition edits on execution feedback [7, 13, 18, 23, 24]; EDA agents have likewise translated design goals into tool actions and scripts [11, 16]. TAPCO combines this program prior with explicit sequential feedback: the LLM proposes configuration or policy changes, while measured history and the physical-design evaluator guide and retain progress.

2.3 Formal Problem Statement

Consider a placed gate-level design d with initial physical state x_d^0. Let C denote the optimizer policy and let θ collect its exposed numeric and categorical controls. Executing a candidate optimizer on the initial state produces

$$x_d = A_{C,\theta}(x_d^0), \tag{1}$$

where x_d is the resulting netlist and placement.

Let $T_d \le 0$ denote setup total negative slack (TNS), and let $P_{d,\mathrm{dyn}}$ and $P_{d,\mathrm{leak}}$ denote dynamic and leakage power. We measure normalized improvements over the initial state as

$$\Delta T_d = \frac{T_d - T_d^0}{|T_d^0|}, \tag{2}$$

$$\Delta P_{d,x} = \frac{P_{d,x}^0 - P_{d,x}}{P_{d,x}^0}, \quad x \in \{\mathrm{dyn}, \mathrm{leak}\}. \tag{3}$$

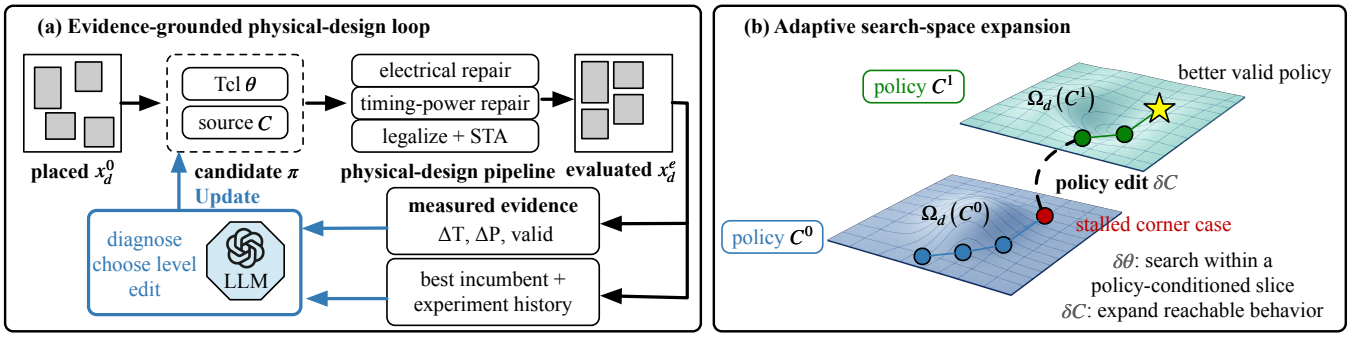


Figure 2: TAPCO overview. (a) A bounded optimizer edit configures a concrete physical-design repair pipeline; measured timing, power, diagnostics, and validity return to the LLM-guided proposal loop. (b) Parameter edits remain within $\Omega_d(C)$, whereas a policy edit opens a new slice when the current interface cannot express the required behavior.

A domain evaluator combines improvements with physical costs:

$$S_d(x_d) = w_T \Delta T_d + w_D \Delta P_{d,\text{dyn}} + w_L \Delta P_{d,\text{leak}} - \sum_j \lambda_j p_{d,j}, \quad (4)$$

where $p_{d,j}$ represents electrical, runtime, displacement, or routing penalties. Placement legality and logic equivalence define the hard predicate $\text{valid}(x_d)$.

Conventional tuning holds C fixed and searches only θ. TAPCO searches for a design-specific configuration and policy:

$$\begin{aligned} (C_d^\star, \theta_d^\star) &= \arg\max_{C,\theta} S_d\big(A_{C,\theta}(x_d^0)\big), \\ \text{s.t.} \quad \text{valid}\big(A_{C,\theta}(x_d^0)\big) &= 1. \end{aligned} \quad (5)$$

Configuration specifies margins, budgets, utilization caps, coverage, and enabled moves; policy specifies scheduling, effort allocation, partial-gain retention, and stagnation recovery.

Equation (5) is solved independently for each design. TAPCO does not select one candidate by optimizing across the benchmark suite; the two suite-level aggregation schemes are reporting functions over final per-design results and are specified in Section 4.

3 TAPCO Framework

Section 2 formulates design-specific optimizer adaptation over a repair policy C and its exposed configuration θ. Solving this problem requires more than selecting another point in a fixed numeric domain: the useful search space may itself depend on what the optimizer learns from a design. TAPCO realizes this idea as the hierarchical, evidence-conditioned process in Fig. 2. The LLM supplies a program prior for diagnosing the current bottleneck and proposing a bounded optimizer edit; the physical-design pipeline supplies the measured outcome that decides whether the edit is useful.

The process alternates among three roles. A *proposal step* converts accumulated evidence into a hypothesis and a configuration- or policy-level modification. An *execution step* applies the resulting optimizer to the original placed design through a concrete repair pipeline. An *update step* retains the best valid optimizer while adding every evaluated attempt to the search history. This separation allows the LLM to reason over source code and design diagnostics without assigning scores to its own proposals.

3.1 Hierarchical Adaptation Space

Let $\pi = (C, \theta)$ denote a candidate optimizer, and let $\Theta_d(C)$ be the configurations exposed by policy C for design d. A fixed policy induces the parameter-search slice

$$\Omega_d(C) = \{(C, \theta) \mid \theta \in \Theta_d(C)\}, \qquad \Omega_d^{\text{TAPCO}} = \bigcup_{C \in \mathfrak{C}_d} \Omega_d(C), \quad (6)$$

where $\mathfrak{C}_d$ is the set of optimizer policies reachable from the initial implementation. Conventional parameter tuning stays within $\Omega_d(C^0)$. TAPCO can additionally change C, which changes the decision logic of the optimizer and may expose a different configuration domain $\Theta_d(C)$. A policy edit therefore opens a new search slice; it is not equivalent to widening the range of an existing parameter.

The hierarchy contains three conceptually distinct layers. At the bottom, a physical-design kernel provides transformations such as sizing, VT substitution, buffering, cloning, and pin swapping. Above it, the *parameter policy* determines the repair envelope through timing and electrical margins, resource budgets, utilization limits, coverage, and move switches. The *source policy* determines how that envelope is consumed: which endpoints are revisited, how moves are ordered, how effort varies with violation severity, and when partial progress is committed or rolled back. These layers are coupled—a source policy defines how a budget is used, whereas its Tcl-visible configuration determines which mechanisms and how much physical resource are available in a particular run.

TAPCO searches this hierarchy from the inside out. When a diagnosed bottleneck maps cleanly to an existing control, a parameter edit changes only θ and preserves the current policy. Such edits are inexpensive, easy to attribute, and sufficient for designs whose desired operating point is already reachable. A policy edit is considered when repeated observations indicate that the missing behavior is structural or temporal rather than scalar. The ordering is a search strategy, not a restriction: after entering a new policy slice, TAPCO again specializes its parameters before deciding whether further expansion is warranted.

3.2 Evidence-Conditioned LLM Proposal

At iteration k, TAPCO distinguishes the optimizer under investigation π_d^k from the best valid incumbent. The latter is stored together

with its evaluated physical result:

$$I_d^k = \left(\pi_{d,b}^k, x_{d,b}^k, S_d(x_{d,b}^k)\right), \qquad x_{d,b}^k = A_{\pi_{d,b}^k}(x_d^0). \tag{7}$$

Executing a candidate produces a structured evaluator observation

$$z_d^k = \left[\Delta T, \Delta P_{\text{dyn}}, \Delta P_{\text{leak}}, \{p_j\}, v\right]_d^k, \tag{8}$$

where v records legality and equivalence. TAPCO complements this score decomposition with a diagnostic record g_d^k: endpoint slacks and their changes, electrical violations, applied-move summaries, placement or equivalence failures, build status, and selected log excerpts. The history H_d^k associates each attempted optimizer with its measured observation and diagnostic record. Consequently, a failed or regressing candidate remains useful evidence even though it cannot replace the incumbent.

A prompt constructor Φ turns this information into an explicit decision context. The prompt first states the optimization objective and hard validity conditions so that a timing gain is not considered independently of power and physical cost. It then presents the incumbent as the reference point, summarizes recent attempts and their outcomes, and exposes the configuration and source regions that may be modified in the current iteration. Finally, it requests one bounded experiment rather than a collection of interacting changes. This last constraint improves credit assignment: the next observation can be related to a concrete hypothesis instead of an uninterpretable bundle of edits.

The proposal is represented as

$$\begin{gathered}(h_d^k, \ell_d^k, \delta_d^k) = \mathcal{L}_\phi\Big(\Phi(d, \pi_d^k, I_d^k, H_d^k, \mathcal{B}_d^k)\Big), \\ \ell_d^k \in \{\text{parameter}, \text{policy}\},\end{gathered} \tag{9}$$

where h_d^k is a bottleneck hypothesis, ℓ_d^k selects the adaptation level, δ_d^k is the proposed edit, and $\mathcal{B}_d^k$ bounds its scope. The structured response must connect an observed symptom to a proposed cause, compare that explanation with earlier attempts, and state the expected direction of the timing–power tradeoff. For example, broadly distributed negative slack may motivate greater coverage or a different endpoint schedule, whereas timing recovery accompanied by excessive leakage may motivate a more selective move set or a different VT strategy. Repeated legality failure instead points to a tighter utilization or displacement envelope. These are proposal priors, not acceptance rules: only measured execution determines whether the hypothesis was correct.

3.3 Physical-Design Execution Pipeline

The selected edit instantiates a complete candidate optimizer and executes it on the design:

$$\widetilde{\pi}_d^{k+1} = U_{\ell_d^k}(\pi_d^k, \delta_d^k), \qquad \widetilde{x}_d^{k+1} = A_{\widetilde{\pi}_d^{k+1}}(x_d^0). \tag{10}$$

A parameter update changes only θ; a policy update changes C and is first built and checked before physical execution. Every successfully constructed candidate starts from the same placed netlist, constraints, libraries, and initial parasitic state. Replaying from x_d^0 rather than continuing from the previous trial makes outcomes comparable and prevents an unsuccessful experiment from contaminating the next one.

The execution path in Fig. 2(a) makes the proposal concrete. It first initializes the placed design and timing state, then applies electrical repair for slew, capacitance, fanout, and wire-length violations under the candidate envelope. Setup repair subsequently traverses the selected endpoint frontier and invokes the enabled physical transformations according to the candidate scheduling and effort policy. Because sizing, buffering, VT substitution, and related moves change both delay and placement demand, the pipeline refreshes parasitic and timing information as required by the policy rather than treating repair as a static ranking problem. The resulting cells are checked and legalized when necessary, after which final timing, dynamic power, leakage power, and physical penalties are measured on the emitted netlist and placement.

This pipeline returns two complementary forms of feedback. The scalar objective answers whether the candidate improved the target operating point; its decomposition distinguishes timing, dynamic-power, leakage, and penalty contributions. The diagnostic channel explains how the candidate reached that outcome: which endpoints remained critical, which move classes dominated, whether electrical violations persisted, and whether legalization or equivalence invalidated the result. Compilation and runtime failures are mapped to the same observation interface with an invalid status and a diagnostic trace. Thus, parameter and source candidates share one evaluation contract despite different construction costs and failure modes.

3.4 Evaluator-Grounded Sequential Update

After evaluation, the best optimizer is updated only by a valid improvement:

$$\pi_{d,b}^{k+1} = \begin{cases} \widetilde{\pi}_d^{k+1}, & \text{valid}(\widetilde{x}_d^{k+1}) \ \wedge \ S_d(\widetilde{x}_d^{k+1}) > S_d(x_{d,b}^k), \\ \pi_{d,b}^k, & \text{otherwise.} \end{cases} \tag{11}$$

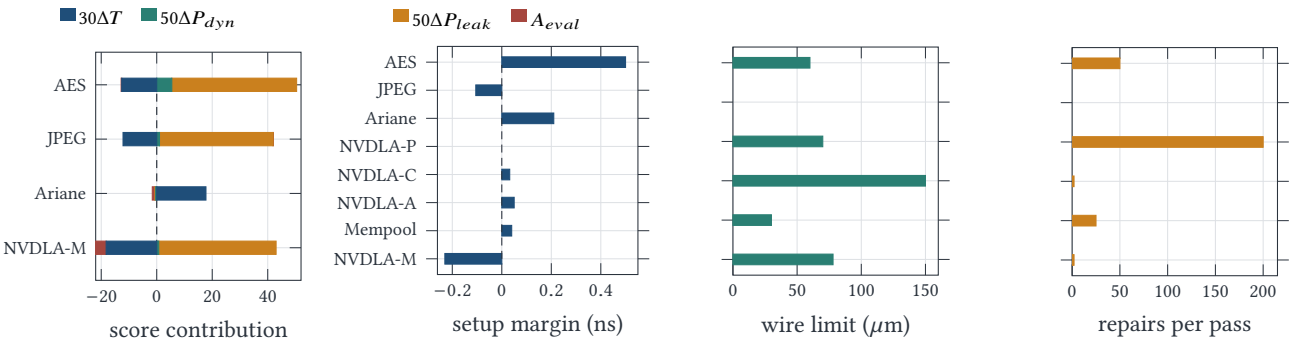
The corresponding result $x_{d,b}^{k+1}$ is retained with the optimizer, so the incumbent score is monotone nondecreasing. Candidate generation, however, does not collapse to greedy hill climbing. Regardless of acceptance, TAPCO appends $(\widetilde{\pi}_d^{k+1}, z_d^{k+1}, g_d^{k+1})$ to H_d^{k+1}. A negative result can rule out an overaggressive margin, reveal that a move class is power-inefficient, or show that a seemingly promising source change breaks legality. The next prompt can therefore avoid repeating the same failure or isolate a smaller variation of the underlying idea.

This feedback organization is Bayesian-inspired in its sequential logic. The LLM's learned knowledge of code and optimization strategies acts as an implicit prior over plausible edits; the evaluator supplies observations from an expensive black box; and each subsequent proposal is conditioned on the incumbent and accumulated measurements. The prompt also makes the exploration–exploitation decision explicit: it asks whether evidence favors refining a known promising configuration or testing a new policy hypothesis. TAPCO does not require a closed-form surrogate, posterior distribution, or acquisition function. Its important property is the division of authority: the proposal mechanism chooses an informative experiment, while the domain evaluator supplies feasibility and objective value and the incumbent rule preserves verified progress.

The search terminates when the allotted development budget is exhausted or the evidence no longer supports a productive bounded edit. The returned solution is the incumbent $\pi_{d,b}$, not the last attempted candidate. Since Eq. (5) is solved separately for each design,

Table 1: Raw QoR comparison on the four designs with valid final-phase results. Power is reported in mW.

	Baseline			TAPCO			
Design	TNS (ns)	P_{dyn}	P_{leak}	TNS (ns)	P_{dyn}	P_{leak}	Score
AES	−12.44	431.904	0.0963	−17.67	384.990	0.0097	37.5818
JPEG	−48.77	293.826	0.1740	−68.51	287.969	0.0315	29.8166
Ariane	−7568.43	640.100	17.900	−3092.71	645.000	18.000	16.1389
NVDLA-M	−13.99	44.539	0.0615	−22.49	43.891	0.0095	20.8050



(a) Score decomposition **(b) Timing frontier** **(c) Electrical-repair reach** **(d) Per-pass effort**

Figure 3: Final-score composition and design-specific scalar controls. Panel (a) shows the weighted timing, power, and evaluator contributions for the four valid final-phase designs; panels (b)–(d) show explicit parameter overrides across all eight optimizers. Blank rows use the inherited setting reported as Default in Table 2.

histories and stopping behavior may differ across the suite without coupling their per-design objectives.

3.5 Adaptive Policy-Space Expansion

The choice of ℓ_d^k determines whether an iteration remains within the current policy-conditioned slice or expands the reachable behavior, as illustrated in Fig. 2(b). Within-slice search changes quantities such as timing margins, TNS coverage, repair budgets, utilization caps, and enabled moves. These variables can substantially shift the operating point, and TAPCO continues to use them whenever the observed bottleneck is expressible by the existing interface.

Escalation is triggered by evidence of an expressivity gap rather than by a fixed iteration count. Typical patterns include parameter saturation with the same endpoints repeatedly unrepaired; effort concentrated on shallow violations while deeper paths dominate the objective; useful partial gains discarded by an all-or-nothing commit rule; or a repair order that repeatedly spends an expensive move before cheaper alternatives are attempted. In each case, a larger scalar budget would repeat the current behavior more aggressively without changing the decision that caused stagnation.

A policy edit addresses the implicated decision and creates a new C^{k+1}. The main expansion dimensions are endpoint selection and reordering, violation-aware effort allocation, checkpoint and partial-progress handling, stagnation recovery, and repair-operator ordering or composition. After such an edit, its Tcl controls remain part of the candidate: the source policy specifies how repair resources are allocated, while the parameter policy specializes how much of each resource the current design can tolerate. This coupling is why TAPCO searches the union in Eq. (6) rather than treating source evolution and hyperparameter tuning independently.

The resulting trajectory adapts its own search resolution. Easy designs may converge entirely inside $\Omega_d(C^0)$; difficult corner cases may cross into a new policy slice and then resume efficient parameter specialization there. In both cases, the same physical pipeline, objective decomposition, and validity gate ground the search. TAPCO thereby enlarges the optimization space only when measured behavior indicates that the fixed interface is the limiting factor.

4 Experiments

This section evaluates TAPCO on the eight-design MLCAD 2026 benchmark. We report overall QoR and examine the parameter and source-policy adaptations behind the results.

4.1 Experimental Setup

We evaluate TAPCO on the eight-design MLCAD 2026 timing-optimization benchmark in the ASAP7 predictive 7-nm technology [4, 8]. The evaluator instantiates Eq. (4) with $(w_T, w_D, w_L) = (30, 50, 50)$ and adds electrical, runtime, displacement, and routing adjustments. The suite contains AES, JPEG, Ariane, four NVDLA variants (P, C, A, and M), and Mempool. Placement legality and logical equivalence are hard constraints.

Candidates use OpenROAD Resizer variants and are evaluated with OpenROAD/OpenSTA for parasitic estimation, incremental

Table 2: Key design-specific parameter settings. Default denotes the inherited setting of the common repair flow.

Design	Wire limit (μm)	Slew/cap. margin (%)	Setup margin (ns)	Repairs/pass
AES	60	10/10	0.500	50
JPEG	Default	Default	−0.105	Default
Ariane	70	Default	0.210	200
NVDLA-P	150	19/19	Default	2
NVDLA-C	30	20/20	0.032	25
NVDLA-A	78	9/9	0.050	2
Mempool	Default	Default	0.040	Default
NVDLA-M	Default	Default	−0.230	Default

timing analysis, repair, and legalization. The physical transformations follow the team's earlier post-placement flow and HyPAS repair foundation [1, 12, 15, 22]. Optimizer development used OpenAI Codex 5.5. We examine overall QoR, design-specific parameter policies, and source-level extensions beyond the exposed interface.

4.2 Overall Results and Score Composition

TAPCO placed second on the official leaderboard, with reported aggregate scores of 40.3031 and 47.5586 [5]. Table 1 compares the four valid final-phase results with the contest baseline.

Figure 3(a) exposes two operating regimes. AES, JPEG, and NVDLA-M are leakage oriented: their leakage reductions contribute 45.0, 40.9, and 42.3 points, respectively, more than compensating for TNS losses of 12.6, 12.1, and 18.2 points. Ariane is timing oriented; its timing recovery contributes 17.7 points while its two power changes together cost less than one point. A timing-only rule would reject three beneficial joint timing–power solutions.

The evaluator adjustment is also design dependent. It reduces the NVDLA-M and Ariane scores by 3.98 and 0.94 points, respectively, but changes AES by only 0.22 points; JPEG receives a small positive adjustment. This decomposition tells TAPCO whether a score change originates in timing, power, or physical cost instead of collapsing all evidence into one scalar observation.

4.3 Design-Specific Parameter Policies

Table 2 reports representative controls; Default denotes the inherited repair-flow setting. Figure 3(b)–(d) shows a 5× range in wire limit, a 100× range in pass budget, and both positive and negative setup margins, revealing design-specific repair envelopes.

Positive setup margins reserve slack and authorize earlier intervention, whereas negative margins narrow the repair frontier so that already acceptable paths are not modified unnecessarily. The observed range therefore changes not only repair intensity but also which paths and violations are eligible for optimization.

Discrete move sets differ as well. AES restricts buffering, VT swapping, and last-gasp repair; NVDLA-P disables VT swapping, buffering, cloning, buffer removal, and downsizing; NVDLA-C adds targeted VT and pin-swap sweeps; and Ariane retains the broad move set with a large repair budget. Thus TAPCO does not converge to one globally tuned configuration.

The valid designs link these policies to different operating points. Ariane combines a positive setup margin with 200 repairs per pass for timing recovery; JPEG and NVDLA-M use negative margins to preserve leakage-oriented solutions. AES reaches a similar score regime through a positive margin and restricted move set, showing that similar objectives need not induce the same policy.

NVDLA-P and NVDLA-C fail validity checks; NVDLA-A and Mempool come from earlier phases. Table 1 and Fig. 3(a) exclude their QoR, but their controls remain in the policy-range analysis.

Table 3: Source-level adaptation beyond fixed parameters.

Capability	Parameter-space limit	Source-level policy
Dynamic priority	Cannot update endpoint order as timing changes.	Recompute and re-sort the violating frontier.
Adaptive effort	Global budgets ignore violation depth and prior effort.	Allocate by severity; avoid repeated heavy repair.
Retention/recovery	Cannot change commit or stagnation behavior.	Retain partial gains; permit bounded recovery.
Candidate set	Move switches cannot define or order candidates.	Add VT/pin-swap pre-passes; select the equivalent-cell policy.

4.4 Source-Policy Extensions

Parameters control effort and operators, not every internal decision. Table 3 maps the resulting limits to source policies.

The four capabilities alter the control structure of repair rather than merely increasing a margin or budget. Dynamic priority revises endpoint order as timing state evolves. Adaptive effort directs expensive work toward severe remaining violations and avoids repeatedly repairing the same region. Retention and recovery change how intermediate outcomes affect subsequent passes, while candidate construction changes which cells and compound moves are considered in the first place.

The two artifact levels are therefore complementary. Parameters specialize the repair envelope for a design; source policies change how that envelope is consumed through scheduling, state updates, and candidate construction. This hierarchy lets TAPCO remain within a compact parameter space when it is expressive enough and expand the reachable behavior when it is not.

5 Conclusion

TAPCO treats the optimizer, not only its exposed parameters, as the object of design-specific adaptation. It combines an LLM prior with evaluator feedback to specialize repair envelopes and extend source policies when necessary. Each iteration uses score decomposition and diagnostic evidence to choose bounded parameter adaptation or policy-space expansion while retaining only valid improvements.

Independent evaluation on the eight-design MLCAD 2026 ASAP7 suite ranked TAPCO second overall and yielded timing- and leakage-oriented strategies. Strategy variations show that LLM-guided adaptation extends optimization beyond fixed hyperparameters. Objectives require evaluator-grounded parameter–policy combinations.

Acknowledgments

This work was supported in part by the National Science and Technology Council (NSTC) under Grant NSTC 112-2221-E-007-106-MY2, Grant NSTC 113-2640-E-011-003, Grant NSTC 113-2221-E-007-082-MY3, Grant NSTC 114-2221-E-007-125-MY3, Grant NSTC 114-2218-E-007-002, and Grant NSTC 114-2221-E-007-126-MY3.

References

[1] T. Ajayi, D. Blaauw, T.-B. Chan, C.-K. Cheng, V. A. Chhabria, D. K. Choo, M. Coltella, S. Dobre, R. Dreslinski, M. Fogaça, et al. 2019. OpenROAD: Toward a Self-Driving, Open-Source Digital Layout Implementation Tool Chain. In *Proc. Government Microcircuit Applications and Critical Technology Conference*. 1105–1110.

[2] C. J. Alpert, M. Hrkić, J. Hu, and S. T. Quay. 2004. Fast and Flexible Buffer Trees that Navigate the Physical Layout Environment. In *Proc. ACM/IEEE Design Automation Conference*. 24–29.

[3] C. J. Alpert, S. K. Karandikar, Z. Li, G.-J. Nam, S. T. Quay, H. Ren, C. N. Sze, P. G. Villarrubia, and M. C. Yildiz. 2007. Techniques for Fast Physical Synthesis. *Proc. IEEE* 95, 3 (2007), 573–599.

[4] ASU VDA Lab. 2026. MLCAD 2026 Contest: Agentic Algorithm Discovery for Timing Optimization. Contest specification. https://github.com/ASU-VDA-Lab/MLCAD26-Contest-Scripts-Benchmarks/blob/main/MLCAD2026-Contest-Algorithm-Discovery.pdf Accessed July 18, 2026.

[5] ASU VDA Lab. 2026. MLCAD26 Contest Results. Online. https://asu-vda-lab.github.io/MLCAD26-Contest/ Accessed July 18, 2026.

[6] C.-P. Chen, C. C. N. Chu, and M. D. F. Wong. 1998. Fast and Exact Simultaneous Gate and Wire Sizing by Lagrangian Relaxation. In *Proc. IEEE/ACM International Conference on Computer-Aided Design*. 617–624.

[7] M. Chen, J. Tworek, H. Jun, Q. Yuan, H. P. de Oliveira Pinto, J. Kaplan, H. Edwards, Y. Burda, N. Joseph, G. Brockman, et al. 2021. Evaluating Large Language Models Trained on Code. *arXiv preprint arXiv:2107.03374* (2021).

[8] L. T. Clark, V. Vashishtha, L. Shifren, A. Gujja, S. Sinha, B. Cline, C. Ramamurthy, and G. Yeric. 2016. ASAP7: A 7-nm FinFET Predictive Process Design Kit. *Microelectronics Journal* 53 (2016), 105–115. doi:10.1016/j.mejo.2016.04.006

[9] S. Do, M. Woo, and S. Kang. 2019. Fence-Region-Aware Mixed-Height Standard Cell Legalization. In *Proc. Great Lakes Symposium on VLSI*. 259–262.

[10] H. Geng, T. Chen, Y. Ma, B. Zhu, and B. Yu. 2023. PTPT: Physical Design Tool Parameter Tuning via Multi-Objective Bayesian Optimization. *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems* 42, 1 (2023), 178–189. doi:10.1109/TCAD.2022.3167858

[11] Z. He, H. Wu, X. Zhang, X. Yao, S. Zheng, H. Zheng, and B. Yu. 2023. ChatEDA: A Large Language Model Powered Autonomous Agent for EDA. In *Proc. ACM/IEEE Workshop on Machine Learning for CAD*. 1–6. doi:10.1109/MLCAD58807.2023.10299852

[12] ISPD 2026 Contest Organizers. 2026. ISPD 2026 Contest: Post-Placement Buffering and Sizing. Contest specification. https://github.com/ABKGroup/ISPD26-Contest Accessed July 18, 2026.

[13] C. E. Jimenez, J. Yang, A. Wettig, S. Yao, K. Pei, O. Press, and K. Narasimhan. 2024. SWE-bench: Can Language Models Resolve Real-World GitHub Issues?. In *Proc. International Conference on Learning Representations*.

[14] D. R. Jones, M. Schonlau, and W. J. Welch. 1998. Efficient Global Optimization of Expensive Black-Box Functions. *Journal of Global Optimization* 13, 4 (1998), 455–492. doi:10.1023/A:1008306431147

[15] F. Liu, W. Tang, B.-Y. Wang, A.-C. Shen, H.-W. Tsao, Y. Ye, Y. Shao, C.-Y. Wang, and B. Yu. 2026. HyPAS: A Hybrid Optimization Framework for Placement and Sizing Co-Optimization. In *Proc. ACM/IEEE Design Automation Conference*. doi:10.1145/3770743.3804137

[16] M. Liu, T.-D. Ene, R. Kirby, C. Cheng, N. Pinckney, R. Liang, J. Alben, H. Anand, S. Banerjee, I. Bayraktaroglu, et al. 2023. ChipNeMo: Domain-Adapted LLMs for Chip Design. *arXiv preprint arXiv:2311.00176* (2023).

[17] T. Luo, D. Newmark, and D. Z. Pan. 2008. Total Power Optimization Combining Placement, Sizing and Multi-VT Through Slack Distribution Management. In *Proc. Asia and South Pacific Design Automation Conference*. 352–357.

[18] N. Shinn, F. Cassano, A. Gopinath, K. Narasimhan, and S. Yao. 2023. Reflexion: Language Agents with Verbal Reinforcement Learning. In *Advances in Neural Information Processing Systems*, Vol. 36.

[19] J. Snoek, H. Larochelle, and R. P. Adams. 2012. Practical Bayesian Optimization of Machine Learning Algorithms. In *Advances in Neural Information Processing Systems*, Vol. 25.

[20] A. Stefanidis, D. Mangiras, C. Nicopoulos, D. Chinnery, and G. Dimitrakopoulos. 2021. Autonomous Application of Netlist Transformations Inside Lagrangian Relaxation-Based Optimization. *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems* 40, 8 (2021), 1672–1686. doi:10.1109/TCAD.2020.3025541

[21] C. N. Sze, C. J. Alpert, J. Hu, and W. Shi. 2005. Path-Based Buffer Insertion. In *Proc. ACM/IEEE Design Automation Conference*. 509–514.

[22] The OpenROAD Project. 2026. Resizer: Timing Optimizations and Design Repairs. OpenROAD documentation. https://openroad.readthedocs.io/en/latest/main/src/rsz/README.html Accessed July 18, 2026.

[23] J. Yang, C. E. Jimenez, A. Wettig, K. Lieret, S. Yao, K. Narasimhan, and O. Press. 2024. SWE-agent: Agent-Computer Interfaces Enable Automated Software Engineering. In *Advances in Neural Information Processing Systems*, Vol. 37. doi:10.52202/079017-1601

[24] S. Yao, J. Zhao, D. Yu, N. Du, I. Shafran, K. Narasimhan, and Y. Cao. 2023. ReAct: Synergizing Reasoning and Acting in Language Models. In *Proc. International Conference on Learning Representations*.

[25] Y. Ye, P. Xu, L. Ren, T. Chen, H. Yan, B. Yu, and L. Shi. 2025. Learning-Driven Physically Aware Large-Scale Circuit Gate Sizing. *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems* 44, 5 (2025), 1901–1914. doi:10.1109/TCAD.2024.3488577

[26] S. Zheng, H. Geng, C. Bai, B. Yu, and M. D. F. Wong. 2023. Boosting VLSI Design Flow Parameter Tuning with Random Embedding and Multi-Objective Trust-Region Bayesian Optimization. *ACM Transactions on Design Automation of Electronic Systems* 28, 5 (2023), 74:1–74:23. doi:10.1145/3597931

A Artifact Appendix

A.1 Abstract

The TAPCO artifact contains the frozen per-design OpenROAD optimizers submitted by Team 14 to the MLCAD 2026 timing-optimization contest, organizer-generated alpha and beta result records, final Tcl and source snapshots, and validation scripts for the quantitative claims in this paper. The mandatory evaluation uses a prebuilt Linux/AMD64 Docker image. It reproduces the AES optimizer output and verifies the four valid beta-3 results, score decomposition, validity boundaries, and design-specific policies. It does not require a GPU, commercial EDA license, or LLM API.

A.2 Artifact Checklist

- **Program:** per-design OpenROAD timing–power optimizers.
- **Binary:** frozen Linux/AMD64 binaries in Docker image `tapco-ae:1.0`.
- **Data set:** eight MLCAD 2026 ASAP7 benchmark designs; all benchmark inputs required for evaluation are included in the Docker image.
- **Run-time environment:** Ubuntu 24.04 Docker image.
- **Hardware:** AMD64 CPU; 32 GB RAM and 80 GB free disk recommended; no GPU required.
- **Execution:** mandatory AES end-to-end run and cached-result validation; optional full run of four valid beta-3 optimizers.
- **Metrics:** TNS, dynamic and leakage power, evaluator penalties, validity status, and final score.
- **Expected results and tolerance:** deterministic fingerprints of the generated AES outputs match the frozen references exactly; workbook-derived numerical terms use an absolute tolerance of 10^{-4}.
- **LLM access:** historical search used OpenAI Codex with model ID `gpt-5.5`; the AE workflow evaluates frozen optimizers and makes no external API call.
- **Zenodo DOI:** `10.5281/zenodo.21566683`.

A.3 Access and Installation

The Zenodo record contains the public source, Tcl snapshots, organizer result records, validation scripts, and exact access metadata for the large Docker archive. The archive itself is provided through the private MLCAD AE Drive. Its filename, image tag, and SHA-256 checksum identify the evaluated version. After downloading both the artifact package and image archive, run:

```
cd ae
bash setup.sh /path/to/tapco-ae-image.tar.gz
```

A.4 Evaluation and Expected Results

The mandatory functional evaluation is:

```
bash run.sh smoke
bash run.sh verify
```

The first command executes the complete AES optimizer in a disposable, network-disabled container, checks the generated DEF and Verilog against the frozen valid beta-3 outputs, and runs the public physical-cell, macro, I/O, and flip-flop checks. The second command verifies the four valid beta-3 scores (AES, JPEG, Ariane, and NVDLA-M), recomputes the score decomposition shown in the paper, checks the invalid and earlier-phase result boundaries, and verifies all per-design Tcl policies. Successful execution prints two `PASS` lines. A longer run of all four valid optimizers is available as `bash run.sh full`, but is optional.

The final contest score contains machine-dependent runtime penalties. Accordingly, cached organizer-derived quantities are recomputed and checked within the stated tolerance, while live execution is checked using deterministic output fingerprints and raw output generation.

A.5 Validity and LLM Scope

The public flow checks physical placement constraints. The contest organizer performed an additional logic-equivalence check; the retained organizer workbook status is authoritative for that check, and reviewers do not need a commercial equivalence tool. The historical LLM search is also outside the mandatory rerun: AE validates the deterministic optimizer artifacts produced by that process. The archived `LLM_PROVENANCE.md` records the exposed model settings, sanitized actual prompt excerpts, and representative proposal-to-measurement traces.