NEVER STEP BACK: MONOTONIC LLM OPTIMIZATION FOR CIRCUIT LAYOUT GENERATION

Anonymous authors
Paper under double-blind review

ABSTRACT

Recent advances in large language models (LLMs) suggest strong potential for automating analog layout design. Yet existing iterative approaches suffer from non-monotonic refinement, where unsuccessful constraint updates may degrade previously verified layouts and propagate errors across subsequent iterations. Moreover, limited reuse of verified design experience across tasks leads to repeated failures and ineffective design exploration. To address these challenges, we propose **MONA**, a training-free multi-agent framework for monotonic analog layout optimization. MONA integrates Adaptive Layout Search (ALS) with Verification-Aware Memory (VAM) and an open-source EDA flow. ALS explores complementary constraint strategies to enable effective design exploration, while VAM employs a Verification Gate to preserve the best deliverable layout and Long-term Memory to distill verified experience for cross-task transfer, accelerating convergence without compromising previously achieved design quality. Across 6 SKY130 analog layout tasks, MONA achieves a 98.9% deliverable-layout success rate, a $2.02\times$ higher success rate than the strongest evaluated LLM-based method, while attaining the highest mean FoM across all tasks. Moreover, MONA reduces LLM tokens per deliverable layout by 53.2%, demonstrating reliable and resource-efficient analog layout optimization.

1 INTRODUCTION

Analog layout remains an iterative process that requires expert guidance and repeated post-layout evaluation to meet demanding electrical specifications. Translating a sized schematic into a physical implementation requires coordinated decisions on device placement, matching, symmetry, and routing, while accounting for layout-induced parasitics (Xu et al., 2019; Lampaert et al., 1995). Unlike digital design, analog layout still relies heavily on handcrafted constraints and simulation-driven optimization. Conventional electronic design automation (EDA) techniques have made significant progress in automating placement, routing, and netlist-to-layout generation, with systems such as ALIGN and MAGICAL providing integrated layout synthesis capabilities (Kunal et al., 2019; Xu et al., 2019). However, deriving effective layout constraints from circuit structure and electrical requirements remains challenging and often requires substantial designer expertise (Sapatnekar, 2023). Even when a generated layout passes design rule checking (DRC) and layout-versus-schematic (LVS) verification, layout-induced parasitics may still degrade its electrical performance. Therefore, a deliverable layout must pass DRC and LVS, yield valid post-layout simulation results, and satisfy all target electrical specifications. This makes efficient post-layout refinement critical for practical analog layout automation under tightly limited EDA budgets and costly simulation cycles.

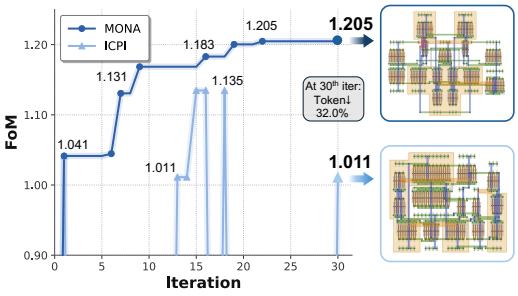


Figure 1: **Illustration of non-monotonic refinement on the StrongARM task.** MONA preserves the best deliverable layout and achieves higher FoM (1.205 vs. 1.011), fewer LLM tokens (148.2K vs. 218.0K), shorter EDA time (3.4 vs. 7.2 min), and smaller area (604.6 vs. 923.3 $\mu\mathrm{m}^2$).

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

Recent advances in large language models (LLMs) have created new opportunities for analog layout automation. By leveraging their capabilities in structured reasoning, constraint generation, and feedback-driven adaptation, LLMs offer a flexible alternative to conventional workflows that rely heavily on handcrafted constraints and numerical optimization (Liu et al., 2025; Chen et al., 2024; Zhang et al., 2026a; Liu et al., 2026). Despite this progress, existing LLM-based layout refinement remains constrained by two key challenges. **(1) Non-Monotonic Refinement.** As LLMs iteratively refine a design, layout quality is not guaranteed to improve (Madaan et al., 2023; Shinn et al., 2023) from one iteration to the next. Constraint updates that fail physical verification or degrade post-layout performance may be retained, causing previously achieved design quality to be lost. As illustrated in Figure 1, such regressive updates can propagate across subsequent iterations, leading to repeated downstream failures. **(2) Ineffective design exploration.** Under a fixed evaluation budget, redundant or ineffective proposals consume substantial EDA and LLM resources, limiting design-space exploration and slowing convergence toward deliverable layouts. These challenges highlight the need for efficient candidate exploration, verification-aware state updates, and systematic reuse of validated experience to improve the reliability and resource efficiency of analog layout refinement.

In this paper, we introduce **MONA**, a training-free multi-agent framework for monotonic analog layout optimization under a limited EDA budget. MONA integrates **Adaptive Layout Search (ALS)** with **Verification-Aware Memory (VAM)** and an open-source EDA flow to enable efficient exploration without regressing from previously verified designs. Specifically, ALS coordinates 3 agents: the Topology Analyst analyzes circuit topology to identify circuit structures and constraint families, the Failure Analyst interprets EDA diagnostics and post-layout measurements to identify failures and potential repair directions, and the Constraint Planner generates 3 candidate constraint sets through exploitation, diversification, and memory-independent exploration to enable effective design exploration. VAM complements this search process with 2 mechanisms. (1) The Verification Gate accepts candidate constraint sets only when they satisfy physical verification requirements and improve the retained design, ensuring that unsuccessful candidates cannot degrade the committed layout. (2) The Memory Distiller transforms evaluation feedback into reusable design knowledge, while Short-term Memory retains current-task experience and Long-term Memory preserves reusable, verified knowledge across tasks to avoid repeated failures and accelerate convergence. Together, ALS and VAM enable a verification-driven optimization process that preserves achieved design quality while improving the reliability and resource efficiency of analog layout generation. In summary, the main contributions of this work are:

1. **Monotonic analog layout optimization.** We formulate analog layout refinement as a monotonic constraint optimization problem under a fixed EDA evaluation budget, emphasizing verification-aware state updates that preserve achieved design quality and prevent non-monotonic refinement.
2. **Adaptive search with verification-aware memory.** We introduce MONA, a training-free multi-agent framework that integrates ALS and VAM. ALS enables effective design exploration through complementary constraint strategies, while the Verification Gate prevents non-monotonic constraint updates and Long-term Memory preserves reusable, verified design knowledge to support cross-task transfer, avoid repeated failures, and accelerate convergence.
3. **Reliable and resource-efficient layout optimization.** We evaluate MONA against strong baselines across 6 analog layout tasks. Results demonstrate monotonic improvement in retained layout quality, a $2.02\times$ higher deliverable-layout success rate than the strongest evaluated LLM-based method, and resource efficiency with 53.2% fewer LLM tokens per deliverable layout.

2 Related Work

2.1 Traditional Methods

Conventional analog layout automation has progressed through constraint-driven generation, physical optimization, and simulation-based search. Integrated generators such as ALIGN and MAGICAL combine circuit recognition, device generation, placement, and routing to automate the transformation from netlists to layouts (Kunal et al., 2019; Dhar et al., 2020; Xu et al., 2019; Chen et al., 2020). At the algorithmic level, simulated annealing, analytical optimization, and mathematical programming have been widely studied for analog placement (Lampaert et al., 1995; Xu et al., 2017; Lin et al., 2022; Grus & Hanzálek, 2022), while specialized routing methods incorporate geometric

and electrical constraints (Ou et al., 2013; Zhang et al., 2023; 2024). Learning-based approaches further introduce performance prediction and differentiable optimization into placement (Li et al., 2020; Xu et al., 2026), while Bayesian optimization and learned candidate filtering improve the efficiency of simulation-based design exploration (Liu et al., 2020; Hakhamaneshi et al., 2019). Despite this progress, translating circuit-level electrical requirements into effective layout constraints remains challenging (Sapatnekar, 2023). Generator-based flows depend on the quality of inferred or designer-provided constraints, while optimization methods are limited by the parameterization and objectives exposed to the search. Reducing the number of evaluations also does not remove the need for physical verification and post-layout simulation, since candidates that appear promising under geometric objectives or learned predictors may still fail DRC, LVS, or electrical specifications. These limitations motivate iterative layout refinement grounded in circuit context, physical verification, and post-layout feedback.

2.2 LLM-Based Methods

LLM-based analog design methods span circuit optimization, layout generation and interaction, and iterative agentic workflows. For circuit optimization, AnaFlow uses specialized agents for circuit interpretation and iterative sizing (Ahmadzadeh et al., 2025), while LEDRO, TopoSizing, and ADO-LLM incorporate LLM guidance into search-space refinement, topology-aware optimization, and candidate generation (Kochar et al., 2025; Wei et al., 2026; Yin et al., 2024). For layout generation and interaction, LayoutCopilot translates natural-language instructions into executable layout operations (Liu et al., 2025), GLayout combines a textual layout representation with fine-tuning and retrieval to generate layout code (Hammoud et al., 2024), and LLANA integrates LLM guidance with Bayesian optimization for layout constraint generation (Chen et al., 2024). More autonomous workflows include PANDA for topology synthesis, sizing, and layout generation (Zhang et al., 2026a), AnalogMaster for connecting schematic-image interpretation with downstream design stages (Qin et al., 2026), and AMS-IO-Agent for structured analog and mixed-signal I/O layout generation (Zhang et al., 2026b). ICPI further performs simulation-aware layout refinement using a structured layout state, sequential parameter updates, and a design journal containing outcomes and reflections (Liu et al., 2026). Although ICPI incorporates failure handling and cross-round experience reuse, it does not establish a monotonic acceptance criterion over post-layout design quality or explicitly transfer verified knowledge across different tasks. In contrast, MONA integrates Adaptive Layout Search (ALS) and Verification-Aware Memory (VAM) with an open-source EDA flow. Its Verification Gate prevents regressive updates to the retained design, while Long-term Memory distills and transfers verified experience across tasks to reduce repeated failures and accelerate convergence. Together, these mechanisms enable monotonic analog layout optimization while improving reliability and resource efficiency under a fixed EDA evaluation budget.

3 Methodology

3.1 Problem Formulation

We formulate analog layout refinement as a budget-constrained optimization problem with physical and electrical verification. Each task is defined by $\mathcal{I} = (G, P, B, S)$, where G is a circuit netlist, P is the process design kit, B is the simulation testbench, and S is the set of electrical performance specifications. The circuit topology and device sizes remain fixed throughout refinement. Given $\mathcal{I}$, the objective is to identify a set of layout constraints x, including placement, matching, symmetry, and routing constraints (Appendix B.2), that produces a high-quality post-layout implementation. For each candidate x, the EDA flow generates a layout L_x and evaluates it through design rule checking (DRC), layout-versus-schematic (LVS) verification, parasitic extraction, and post-layout simulation. The evaluation returns post-layout performance metrics m_x, such as gain, unity-gain bandwidth (UGB), common-mode rejection ratio (CMRR), and phase margin (PM), together with diagnostic feedback from the EDA tools. We define the deliverability of a candidate as

$$D(x) = \begin{cases} 1, & \text{if DRC and LVS pass, simulation is valid,} \\ & \text{and } m_x \text{ satisfies all specifications in } S, \\ 0, & \text{otherwise.} \end{cases} \tag{1}$$

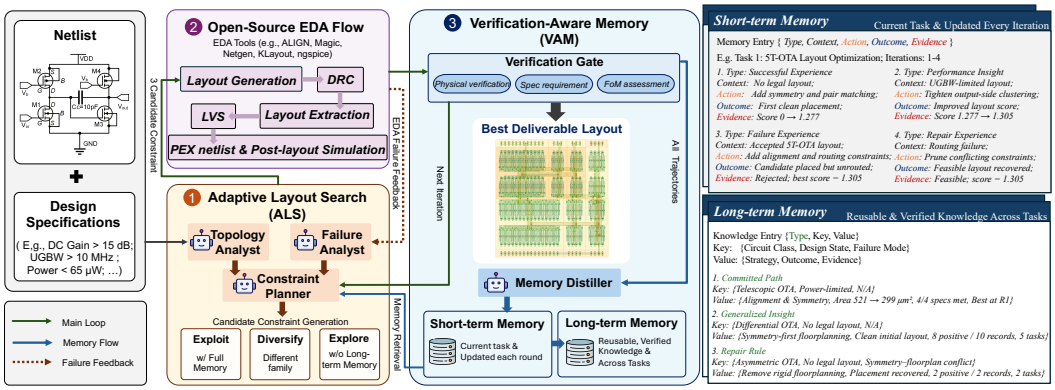


Figure 2: **MONA Framework Overview.** MONA integrates Adaptive Layout Search (ALS) and Verification-Aware Memory (VAM) with an open-source EDA flow. ALS generates candidate constraint sets based on circuit topology and EDA feedback, while VAM employs a Verification Gate to preserve the best deliverable layout and distills evaluation outcomes into short-term and long-term memory. Together, these components ensure monotonic refinement, support cross-task knowledge reuse, and improve optimization efficiency under a limited EDA budget.

Under a budget of N EDA evaluations, the goal is to find a deliverable layout that maximizes the FoM F:

$$x^* \in \underset{\substack{x \in \{x_1,\ldots,x_N\} \\ D(x)=1}}{\arg\max} \; F(L_x, m_x), \tag{2}$$

where F combines post-layout electrical performance and layout area, as defined in Equation 3. Beyond maximizing the FoM, our objective is to ensure monotonic refinement: once a deliverable layout is found, subsequent accepted updates must preserve its deliverability and maintain or improve its FoM. By decoupling candidate exploration in ALS from state updates governed by the Verification Gate, MONA enables exploration while preserving the best verified design.

3.2 Framework Overview

We present MONA, a training-free multi-agent framework for monotonic analog layout optimization that integrates Adaptive Layout Search (ALS) with Verification-Aware Memory (VAM). As shown in Figure 2, ALS coordinates 3 specialized agents: the Topology Analyst, Failure Analyst, and Constraint Planner. The Topology Analyst analyzes circuit topology, while the Failure Analyst interprets EDA diagnostics and post-layout performance. Guided by these insights, the Constraint Planner generates 3 candidate constraint sets through Exploit with full memory, Diversify across different constraint families, and Explore without long-term memory. VAM incorporates a Verification Gate that selectively accepts constraint updates based on physical verification, specification satisfaction, and FoM, preventing regressive updates and preserving the best deliverable layout. Meanwhile, the Memory Distiller consolidates evaluation outcomes into Short-term Memory for within-task refinement and Long-term Memory for cross-task knowledge reuse. The accumulated feedback and retrieved experience guide subsequent exploration, reducing repeated failures and accelerating convergence toward deliverable layouts. Together, ALS and VAM enable monotonic refinement and efficient design exploration without updating the underlying LLM parameters. We detail ALS and VAM in the following subsections. The full procedure is given in Appendix C.

3.3 Adaptive Layout Search (ALS)

Adaptive Layout Search (ALS) coordinates 3 specialized LLM agents: the Topology Analyst, Failure Analyst, and Constraint Planner. Together, they generate candidate layout constraints using circuit-level analysis, EDA feedback, and retrieved experience. Rather than relying on sequential modifications of a single constraint set, ALS combines exploitation of previously verified designs with diversified exploration to generate promising candidates under a limited EDA budget. Candi-

date updates are subsequently evaluated by the EDA flow and selectively accepted by the Verification Gate, separating exploration from committed state updates.

Topology Analyst. Given the sized netlist and design requirements, the Topology Analyst performs circuit-level analysis to identify functional blocks, critical devices and nets, and matching relationships. Based on the circuit topology, it provides initial guidance on placement, symmetry, and routing constraints. This analysis is performed once before iterative refinement and reused by the Constraint Planner for subsequent candidate generation.

Failure Analyst. The Failure Analyst interprets feedback from the EDA flow to diagnose unsuccessful layout attempts. It examines failures encountered during layout generation, DRC, LVS, parasitic extraction, and post-layout simulation. For candidates with valid simulation results, it also identifies unmet electrical specifications. Based on the observed failures, it identifies potential causes and suggests relevant constraint adjustments. These diagnoses provide actionable feedback to the Constraint Planner, allowing subsequent refinement to address observed failures and avoid repeating ineffective modifications.

Constraint Planner. The Constraint Planner integrates circuit analysis, failure diagnosis, and retrieved memory to generate 3 candidate constraint sets through complementary strategies. (1) **Exploit** leverages full memory to refine the retained constraint set using current-task evaluation history and verified design knowledge retrieved from Long-term Memory. It prioritizes adjustments that address physical verification failures, unmet electrical specifications, or improvements in post-layout performance. (2) **Diversify** explores constraint families to avoid repeatedly modifying the same layout parameters. For example, when placement adjustments do not resolve a performance violation, Constraint Planner may explore routing-related constraints. (3) **Explore** generates candidates using current-task information and within-task feedback without retrieving Long-term Memory, providing an exploration path independent of accumulated cross-task knowledge. Together, these strategies balance the exploitation of verified experience with alternative exploration paths, reducing redundant evaluations and supporting efficient convergence toward deliverable layouts.

3.4 VERIFICATION-AWARE MEMORY (VAM)

The 3 candidate constraint sets generated by ALS through Exploit, Diversify, and Explore are evaluated through an open-source EDA flow comprising layout generation, physical verification, parasitic extraction, and post-layout simulation. Each evaluation produces a trajectory recording the proposed constraint changes, DRC and LVS results, post-layout performance when available, failure diagnostics, and the corresponding acceptance decision. As illustrated in Figure 2, Verification-Aware Memory (VAM) comprises a Verification Gate and a Memory Distiller. The Verification Gate selectively retains evaluated designs to ensure monotonic refinement, while the Memory Distiller transforms evaluation trajectories into reusable design knowledge, maintaining Short-term Memory for within-task refinement and Long-term Memory for cross-task knowledge reuse.

Verification Gate. The Verification Gate evaluates candidate layouts based on physical validity, specification satisfaction, and FoM. It checks DRC and LVS results and, when valid post-layout measurements are available, determines whether all electrical specifications are satisfied. Once a deliverable layout is found, subsequent candidates are accepted only if they preserve deliverability and maintain or improve the retained FoM. This ensures that the best deliverable layout is preserved throughout refinement, preventing non-monotonic updates. Both accepted and rejected trajectories remain available to the Memory Distiller, allowing unsuccessful evaluations to inform subsequent exploration without degrading the retained design. Appendix E traces one complete run.

Memory Distiller. The Memory Distiller extracts reusable design knowledge from evaluation trajectories, including proposed constraint changes, physical verification results, post-layout performance, and failure diagnostics from both successful and failed attempts (Zhao et al., 2024; Ouyang et al., 2026). It maintains 2 complementary memory components. **Short-term Memory** records the current task's evaluation history and distilled insights, providing the Constraint Planner with context for subsequent refinement. **Long-term Memory** preserves reusable, verified knowledge across tasks, enabling relevant experience to be retrieved for new circuit designs. By reusing verified design strategies and failure insights, Long-term Memory helps avoid repeated failures and accelerate convergence toward deliverable layouts. Together, these memories support within-task refinement and cross-task knowledge reuse without updating the underlying LLM parameters.

4 EXPERIMENTS

4.1 EXPERIMENTAL SETUP

Benchmarks. We evaluate MONA on 6 fixed-sizing analog layout tasks spanning 4 diverse circuit topologies, drawn from AutoSizer (AMS-SizingBench) and CABAgent (Yu et al., 2026; Hu et al., 2026), as summarized in Table 5. All circuits are implemented in the `sky130A` process and evaluated at the typical process corner with a $1.8\,\mathrm{V}$ supply. Each task uses a fixed transistor-level netlist, isolating layout optimization from circuit sizing. The benchmark size reflects the limited availability of reproducible analog layout tasks: commercial design datasets are generally inaccessible, while many public datasets lack the electrical specifications and testbenches required for post-layout evaluation. For example, ICPI (Liu et al., 2026) evaluates 2 OTA circuits using commercial PDKs and EDA tools, with neither benchmark publicly available. Detailed specifications for all 6 tasks are provided in Appendix A. The EDA flow and tool versions are given in Appendix B.1.

Baselines. We compare MONA with ALIGN (Dhar et al., 2020), BO (Frazier, 2018), and ICPI (Liu et al., 2026). ALIGN generates layouts using the shared physical-design backend without additional layout constraints. BO uses a tree-structured Parzen estimator (Bergstra et al., 2011) to optimize the same constraint parameters available to the agent-based methods, starting with two randomly initialized trials. ICPI refines layouts through its Analyzer, Supervisor, Executor, and Reflector workflow, which we reproduce within the same layout-generation and verification flow (Appendix B.4). Both ICPI and MONA use `GPT-5.4` and `Qwen3-8B`. All methods use identical netlists, specifications, initial layouts, and evaluation functions to ensure a fair comparison.

Metrics. We evaluate success rate, layout quality, and resource consumption. For the main experiments, we conduct 15 independent runs per task and method across 6 tasks, with 5 refinement iterations per run (Appendix B.3). A layout is *deliverable* if it passes DRC and LVS, yields valid post-layout simulation results, and meets all electrical specifications. We define $\mathrm{Success} = (c/n)\times 100\%$, where c is the number of successful runs and $n = 15$ for each task and method. Layout quality combines electrical performance and area:

$$s_j = \begin{cases} r_j^2, & r_j < 1, \\ 2 - r_j^{-1}, & r_j \geq 1, \end{cases} \qquad F(L, m) = \sqrt{s_A \min_j s_j}. \tag{3}$$

Here, r_j measures performance relative to the specification target: $r_j = m_j/\tau_j$ when larger values are preferred and $r_j = \tau_j/m_j$ when smaller values are preferred (Liu et al., 2026). The score applies a quadratic penalty below the target and a bounded reward above it. The area score s_A follows the same rule with $r_A = A_0/A(L)$, where A_0 is the area of the layout generated by ALIGN. Resource consumption is measured by cumulative EDA execution time (s) and total LLM token usage (K).

4.2 EXPERIMENTAL RESULTS

As shown in Table 1, MONA achieves a 98.9% overall deliverable-layout success rate, compared with 48.9% for ICPI and 17.8% for BO, an improvement of 50.0 percentage points over ICPI. ALIGN, which generates a fixed layout without iterative refinement, achieves a 50.0% success rate, producing deliverable layouts on only half of the benchmark tasks. Under the same evaluation budget, MONA achieves success rates of at least 93.3% across all six tasks and the highest mean FoM on every task. These results are consistent with monotonic refinement enabled by Verification-Aware Memory (VAM), which preserves verified improvements, prevents unsuccessful updates from degrading retained layouts, and reuses validated experience to guide subsequent search. Moreover, MONA reduces LLM tokens per deliverable layout from 22.2K to 10.4K relative to ICPI, a 53.2% reduction (Appendix D.1). Together, these results demonstrate that monotonic, verification-aware refinement improves layout reliability and solution quality while reducing LLM cost.

Table 2 compares the post-layout performance across four circuit topologies. ALIGN fails to meet the UGB specification for the telescopic OTA and fails LVS for the current-mirror OTA. BO improves the telescopic OTA FoM from 0.842 to 1.229 relative to ALIGN, but produces larger layouts for the five-transistor and current-mirror OTAs and fails to produce a deliverable StrongARM comparator. In contrast, MONA achieves the highest FoM across all four topologies. Relative to ICPI,

Table 1: **Main results** across 6 tasks with 15 independent runs and 5 iterations per run. *SR*: deliverable-layout success rate (%); *FoM*: mean over deliverable layouts, summed across tasks for Overall. *Direct*: 1 LLM call per iteration without an agentic framework. Best and second-best results are **bolded** and underlined; ALIGN is a reference and excluded from ranking.

Method	Tele-OTA-1p		Tele-OTA-17f		5T-OTA-1p		5T-OTA-30f		CM-OTA		StrongARM		Overall	
	SR	FoM	SR	FoM	SR	FoM	SR	FoM	SR	FoM	SR	FoM	SR	FoM
ALIGN	100.0	1.041	0.0	–	100.0	1.068	0.0	–	0.0	–	100.0	1.043	50.0	3.152
LLM-free														
BO	20.0	1.180	6.7	1.069	33.3	0.906	0.0	–	46.7	1.078	0.0	–	17.8	4.233
Qwen3-8B														
ICPI	46.7	1.127	40.0	1.181	26.7	0.923	6.7	0.897	0.0	–	0.0	–	20.0	4.128
Direct	80.0	1.213	**100.0**	1.209	46.7	0.970	13.3	0.907	6.7	1.091	0.0	–	41.1	5.390
MONA (Ours)	**100.0**	1.237	**100.0**	1.243	**100.0**	1.003	33.3	0.914	93.3	1.125	60.0	1.017	81.1	6.539
GPT-5.4														
ICPI	40.0	1.233	33.3	1.230	80.0	0.997	60.0	0.915	60.0	1.185	20.0	1.135	48.9	6.695
MONA (Ours)	**100.0**	**1.259**	**100.0**	**1.261**	**100.0**	**1.068**	**93.3**	**0.932**	**100.0**	**1.206**	**100.0**	**1.205**	**98.9**	**6.931**
Δ vs. best baseline	+20.0	+0.026	+0.0	+0.031	+20.0	+0.071	+33.3	+0.017	+40.0	+0.021	+80.0	+0.070	+50.0	+0.236

Table 2: **Post-layout performance across 4 circuit topologies.** Post-layout metrics of the layouts produced by ALIGN, BO, ICPI, and MONA. For the OTAs, gain, unity-gain bandwidth (UGB), and power are the specified electrical metrics; for the comparator, delay, output levels, and power. Area denotes the layout area, and FoM denotes the layout quality defined in Equation 3; arrows indicate the preferred direction of each metric, and the best FoM per topology is in bold.

OTA

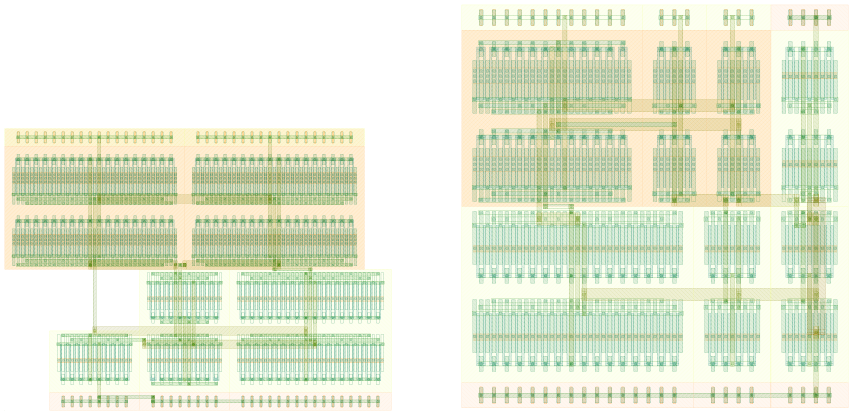
Topology	Method	Gain (dB) ↑	UGB (MHz) ↑	Power (μW) ↓	Area (μm^2) ↓	FoM ↑
Telescopic	ALIGN†	21.16	134.85	59.61	521.2	0.842
	BO	22.77	164.26	59.98	311.3	1.229
	ICPI	22.88	178.07	59.95	310.0	1.231
	MONA	22.73	172.97	59.91	274.2	**1.261**
Five-transistor	ALIGN†	22.31	191.46	55.82	523.6	0.785
	BO	22.16	216.72	56.74	592.9	0.884
	ICPI	22.32	227.68	56.35	586.4	0.915
	MONA	22.28	218.88	55.97	561.7	**0.932**
Current-mirror	ALIGN§	–	–	–	972.3	0.000
	BO	17.000	13.99	57.36	786.0	1.154
	ICPI	16.981	12.95	56.40	725.2	1.183
	MONA	16.976	14.09	57.73	673.5	**1.206**

Comparator

Topology	Method	Delay (ps) ↓	V_{OH} (V) ↑	V_{OL} (V) ↓	Power (μW) ↓	Area (μm^2) ↓	FoM ↑
StrongARM	ALIGN	548.5	1.775	0.00220	245.75	890.8	1.043
	BO‡	–	–	–	–	–	–
	ICPI	527.9	1.795	0.000356	221.63	736.3	1.135
	MONA	503.0	1.797	0.000231	221.28	604.6	**1.205**

Notes. Arrows indicate the preferred direction of each metric. Comparator power denotes average transient power. † ALIGN misses the UGB specification (3/4 specifications met). § ALIGN fails LVS for the current-mirror OTA. ‡ None of the BO candidates produced a comparator layout.

it reduces the telescopic OTA area from 310.0 to 274.2 μm^2 and the current-mirror OTA area from 725.2 to 673.5 μm^2. For the StrongARM comparator, MONA further reduces delay from 527.9 to 503.0 ps and area from 736.3 to 604.6 μm^2, while maintaining comparable power consumption. Together with the higher deliverable-layout success rate and lower token cost in Table 11, these results demonstrate the effectiveness of combining verification-aware state preservation with reusable design experience for reliable, resource-efficient layout refinement. Figure 3 further provides a visual comparison of the current-mirror OTA layouts produced by ALIGN and MONA on the same netlist.



(a) ALIGN (b) MONA

Figure 3: **Current-mirror OTA layout comparison:** MONA achieves a 30.7% area reduction. MONA reduces the layout area from 972.3 to 673.5 $\mu\mathrm{m}^2$ while achieving 16.976 dB gain, 14.09 MHz UGB, and 57.73 μW power, satisfying all specifications. In contrast, ALIGN fails LVS verification and therefore does not constitute a physically valid, deliverable design.

4.3 Ablation Study

To evaluate the contribution of each component, we conduct ablation studies by removing Adaptive Layout Search (ALS), Verification-Aware Memory (VAM), and the Verification Gate, using the same experimental setting as the main experiments (Section 4.1). As shown in Table 3, MONA achieves a 98.9% deliverable-layout success rate, compared with 88.9% without ALS, 61.1% without VAM, and 47.8% without the Verification Gate. Removing the Verification Gate produces the largest reduction in success rate and decreases the aggregate FoM from 6.931 to 3.106. Figure 5 visualizes this effect: without the Verification Gate, the retained FoM repeatedly drops during refinement, whereas MONA preserves the best verified layout and improves monotonically; equipping ICPI with the same Verification Gate likewise makes its refinement monotonic, though without Long-term Memory and the Constraint Planner it converges more slowly and to a lower FoM.

Removing VAM also substantially reduces success rate and aggregate FoM, while removing ALS increases the EDA time per deliverable layout from 186 to 257 seconds. These results demonstrate the complementary contributions of ALS and VAM to reliable and resource-efficient layout optimization. Figure 4 examines the source of the final retained layouts in the ALS component. Across 90 runs, Exploit, Diversify, and Explore contribute 34, 33, and 23 final layouts, with no single strategy dominating (Appendix D.3). This indicates that full-memory exploitation, diversification across constraint families, and exploration without long-term memory each provide useful candidates, supporting effective design exploration while preserving previously achieved design quality.

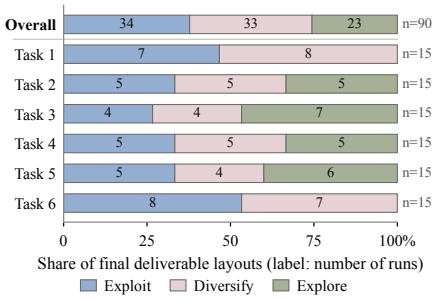


Figure 4: Candidate source of the final deliverable layout (Exploit, Diversify, or Explore) across all evaluation runs.

Method	SR (%)	FoM	EDAs
MONA	**98.9**	**6.931**	**186**
w/o ALS	88.9	6.006	257
w/o VAM	61.1	3.949	194
w/o Gate	47.8	3.106	196

Table 3: **Ablation study.** The effectiveness of ALS, VAM, and the Verification Gate is evaluated by systematically removing individual components of MONA. FoM: sum over the 6 tasks of the mean deliverable-layout FoM; EDAs: EDA seconds per deliverable layout. Per-task results in Appendix D.2.

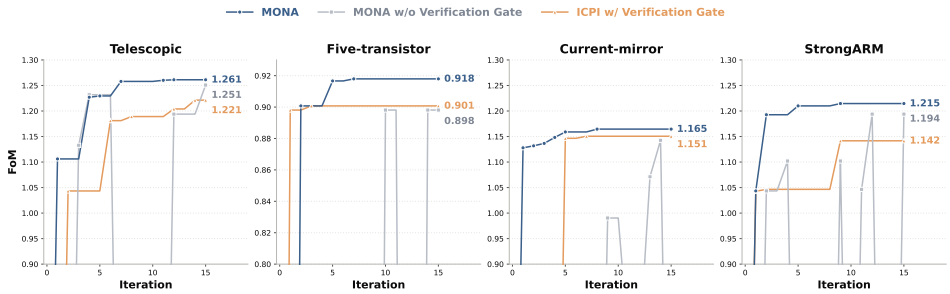


Figure 5: **FoM across refinement iterations** on 4 topologies. MONA achieves monotonic improvement and higher final FoM; ICPI improves monotonically with the same Verification Gate, while removing the gate causes regression. Numbers denote final FoM.

Task	BO			ICPI			**MONA (Ours)**		
	Token	EDA	Runs	Token	EDA	Runs	Token	EDA	Runs
Tele-OTA-1p	–	371	●○○●●	41.9	353	●○●●●	14.1	129	●●●●●
Tele-OTA-17f	–	1127	○●●○○	49.2	330	●●●●●	19.5	104	●●●●●
5T-OTA-1p	–	146	●●●○○	34.7	192	●●●●●	17.9	56	●●●●●
5T-OTA-30f	–	663	●●○○○	32.9	174	●●○●●	65.6	685	●●●●●
CM-OTA	–	867	○●●●●	25.0	170	●●○○●	31.1	477	●●●●●
StrongARM	–	–	○○○○○	87.3	430	○●○○●	11.3	201	●●●●●
Sum	–	3174	14/30	271.0	1649	23/30	159.5	1652	30/30

Table 4: **Cost to the first deliverable layout.** *Token* (K) and *EDA* time (s) are averaged over successful runs. *Runs*: 5 repeats per task (●: success; ○: failure). *Sum*: sum of per-task means and successful runs out of 30. BO uses no LLM; – denotes not applicable or no success.

4.4 Efficiency Analysis

Table 4 compares the cost of reaching the first deliverable layout under a budget of 30 EDA calls per run, with 5 independent repeats per task. MONA successfully obtains a deliverable layout in all 30 runs, compared with 23 for ICPI and 14 for BO, demonstrating reliable convergence under the same evaluation budget. Relative to ICPI, MONA reduces the summed mean LLM token cost to first success from 271.0K to 159.5K, a 41.1% reduction. Figure 6 further illustrates this efficiency on Task 6, showing substantially lower cumulative token usage throughout the refinement process. MONA maintains comparable summed mean EDA time (1652 vs. 1649 seconds) across successful runs while achieving 7 additional successful runs. These results demonstrate that MONA reliably reaches deliverable layouts with substantially lower LLM overhead, supporting resource-efficient analog layout optimization.

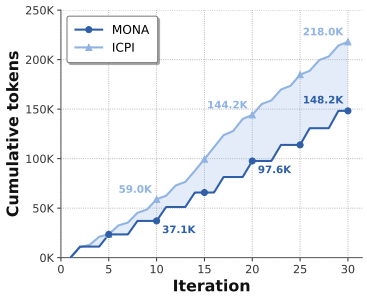


Figure 6: **LLM token usage on Task 6** across refinement iterations. MONA reduces cumulative token usage by 32.0%.

5 Conclusion and Future Work

In this paper, we introduced MONA, a training-free multi-agent framework for monotonic analog layout optimization. By combining Adaptive Layout Search (ALS) with Verification-Aware Memory (VAM), MONA preserves verified design quality while improving exploration efficiency and cross-task knowledge reuse. Across 6 SKY130 tasks, MONA achieves a 2.02× higher deliverable-layout success rate and reduces LLM tokens per deliverable layout by 53.2%. Future work will extend the framework to more diverse circuits and technology nodes and study cross-technology transfer of verified design knowledge, as well as integration with commercial EDA flows and PDKs.

AI USE STATEMENT

This work investigates the use of large language models (LLMs) for analog layout refinement. LLMs are employed as components of the proposed framework and the LLM-based baseline, as described in the Methodology and Experiments sections. Generative AI tools were also used to assist with manuscript drafting and language editing. All AI-assisted content was reviewed and revised by the authors. The authors independently verified the reported experimental results and take full responsibility for the methods, findings, and final content of this paper.

REPRODUCIBILITY STATEMENT

The problem formulation and the algorithm are given in Section 3, and the experimental setup, evaluation metrics, and baseline configurations in Section 4.1. Appendix A documents the six benchmark tasks and their electrical specifications. Appendix B lists the EDA flow and tool versions, the constraint vocabulary, the iteration structure and all hyperparameters, the baseline configurations, and the compute used (Appendix B.5). Appendix C gives the full algorithm in pseudocode and the input and output of every LLM agent; the system prompts are released with the code. Additional results and a complete refinement trajectory are given in Appendices D and E. The code, the benchmark tasks, the baseline adapters, the experiment drivers, and the raw data of every reported run will be released with the camera-ready version (Appendix B.6).

REFERENCES

Mohsen Ahmadzadeh, Kaichang Chen, and Georges Gielen. Anaflow: Agentic llm-based workflow for reasoning-driven explainable and sample-efficient analog circuit sizing. In *2025 IEEE/ACM International Conference On Computer Aided Design (ICCAD)*, pp. 1–7. IEEE, 2025.

James Bergstra, Rémi Bardenet, Yoshua Bengio, and Balázs Kégl. Algorithms for hyper-parameter optimization. *Advances in neural information processing systems*, 24, 2011.

Guojin Chen, Keren Zhu, Seunggeun Kim, Hanqing Zhu, Yao Lai, Bei Yu, and David Z Pan. Llm-enhanced bayesian optimization for efficient analog layout constraint generation. *arXiv preprint arXiv:2406.05250*, 2024.

Hao Chen, Mingjie Liu, Biying Xu, Keren Zhu, Xiyuan Tang, Shaolan Li, Yibo Lin, Nan Sun, and David Z Pan. Magical: An open-source fully automated analog ic layout system from netlist to gdsii. *IEEE Design & Test*, 38(2):19–26, 2020.

Tonmoy Dhar, Kishor Kunal, Yaguang Li, Meghna Madhusudan, Jitesh Poojary, Arvind K Sharma, Wenbin Xu, Steven M Burns, Ramesh Harjani, Jiang Hu, et al. Align: A system for automating analog layout. *IEEE Design & Test*, 38(2):8–18, 2020.

Peter I Frazier. A tutorial on bayesian optimization. *arXiv preprint arXiv:1807.02811*, 2018.

Josef Grus and Zdeněk Hanzálek. Matheuristic local search for the placement of analog integrated circuits. In *International Conference on Operations Research and Enterprise Systems*, pp. 178–200. Springer, 2022.

Kourosh Hakhamaneshi, Nick Werblun, Pieter Abbeel, and Vladimir Stojanović. Bagnet: Berkeley analog generator with layout optimizer boosted with deep neural networks. In *2019 IEEE/ACM International Conference on Computer-Aided Design (ICCAD)*, pp. 1–8. IEEE, 2019.

Ali Hammoud, Chetanya Goyal, Sakib Pathen, Arlene Dai, Anhang Li, Gregory Kielian, and Mehdi Saligane. Human language to analog layout using glayout layout automation framework. In *Proceedings of the 2024 ACM/IEEE International Symposium on Machine Learning for CAD*, pp. 1–7, 2024.

Jinhai Hu, Jiageng Wang, Xinzhe Xie, Zhixuan Bao, Wang Ling Goh, Xiaoli Li, Xinrun Wang, Xun Xu, Zhuoyi Lin, and Yuan Gao. CABAgent: A comprehensive layout-aware analog benchmark generation framework driven by self-evolving LLM agents for analog circuit design automation. `https://github.com/sscs-ose/sscs-ose-code-a-chip.github.`

io/tree/main/VLSI26/accepted_notebooks/CABAgent, 2026. IEEE SSCS Code-a-Chip, VLSI 2026, award-winning notebook.

Dimple Vijay Kochar, Hanrui Wang, Anantha P Chandrakasan, and Xin Zhang. Ledro: Llm-enhanced design space reduction and optimization for analog circuits. In *2025 IEEE International Conference on LLM-Aided Design (ICLAD)*, pp. 141–148. IEEE, 2025.

Kishor Kunal, Meghna Madhusudan, Arvind K Sharma, Wenbin Xu, Steven M Burns, Ramesh Harjani, Jiang Hu, Desmond A Kirkpatrick, and Sachin S Sapatnekar. Align: Open-source analog layout automation from the ground up. In *Proceedings of the 56th Annual Design Automation Conference 2019*, pp. 1–4, 2019.

Koen Lampaert, Georges Gielen, and Willy M Sansen. A performance-driven placement tool for analog integrated circuits. *IEEE Journal of solid-state circuits*, 30(7):773–780, 1995.

Yaguang Li, Yishuang Lin, Meghna Madhusudan, Arvind Sharma, Wenbin Xu, Sachin S Sapatnekar, Ramesh Harjani, and Jiang Hu. A customized graph neural network model for guiding analog ic placement. In *Proceedings of the 39th International Conference on Computer-Aided Design*, pp. 1–9, 2020.

Yishuang Lin, Yaguang Li, Donghao Fang, Meghna Madhusudan, Sachin S Sapatnekar, Ramesh Harjani, and Jiang Hu. Are analytical techniques worthwhile for analog ic placement? In *2022 Design, Automation & Test in Europe Conference & Exhibition (DATE)*, pp. 154–159. IEEE, 2022.

Bingyang Liu, Haoyi Zhang, Xiaohan Gao, Zichen Kong, Xiyuan Tang, Yibo Lin, Runsheng Wang, and Ru Huang. Layoutcopilot: An llm-powered multiagent collaborative framework for interactive analog layout design. *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, 44(8):3126–3139, 2025.

Bingyang Liu, Ziming Wei, Xiaohan Gao, and David Z Pan. Simulation-aware in-context policy improvement for llm-aided analog layout refinement. *arXiv preprint arXiv:2608.13767*, 2026.

Mingjie Liu, Keren Zhu, Xiyuan Tang, Biying Xu, Wei Shi, Nan Sun, and David Z Pan. Closing the design loop: Bayesian optimization assisted hierarchical analog layout synthesis. In *2020 57th ACM/IEEE design automation conference (DAC)*, pp. 1–6. IEEE, 2020.

Aman Madaan, Niket Tandon, Prakhar Gupta, Skyler Hallinan, Luyu Gao, Sarah Wiegreffe, Uri Alon, Nouha Dziri, Shrimai Prabhumoye, Yiming Yang, et al. Self-refine: Iterative refinement with self-feedback. *Advances in neural information processing systems*, 36:46534–46594, 2023.

Hung-Chih Ou, Hsing-Chih Chang Chien, and Yao-Wen Chang. Simultaneous analog placement and routing with current flow and current density considerations. In *Proceedings of the 50th Annual Design Automation Conference*, pp. 1–6, 2013.

Siru Ouyang, Jun Yan, I Hsu, Yanfei Chen, Ke Jiang, Zifeng Wang, Rujun Han, Long Le, Samira Daruki, Xiangru Tang, et al. Reasoningbank: Scaling agent self-evolving with reasoning memory. In *International Conference on Learning Representations*, volume 2026, pp. 94327–94354, 2026.

Xian Rong Qin, Yong Zhang, Ying Hu, Tao Su, Bo-Wen Jia, and Ning Xu. Analogmaster: Large language model-based automated analog ic design framework from image to layout. *arXiv preprint arXiv:2604.20916*, 2026.

Sachin S Sapatnekar. The align automated analog layout engine: Progress, learnings, and open issues. In *Proceedings of the 2023 International Symposium on Physical Design*, pp. 101–102, 2023.

Noah Shinn, Federico Cassano, Ashwin Gopinath, Karthik Narasimhan, and Shunyu Yao. Reflexion: Language agents with verbal reinforcement learning. *Advances in neural information processing systems*, 36:8634–8652, 2023.

Ziming Wei, Zichen Kong, Yuan Wang, David Z Pan, and Xiyuan Tang. Toposizing: An llm-aided framework of topology-based understanding and sizing for ams circuits. *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, 2026.

Biying Xu, Shaolan Li, Xiaoqing Xu, Nan Sun, and David Z Pan. Hierarchical and analytical placement techniques for high-performance analog circuits. In *Proceedings of the 2017 ACM on International Symposium on Physical Design*, pp. 55–62, 2017.

Biying Xu, Keren Zhu, Mingjie Liu, Yibo Lin, Shaolan Li, Xiyuan Tang, Nan Sun, and David Z Pan. Magical: Toward fully automated analog ic layout leveraging human and machine intelligence. In *2019 IEEE/ACM International Conference on Computer-Aided Design (ICCAD)*, pp. 1–8. IEEE, 2019.

Peng Xu, Yuan Pu, Keren Zhu, Tinghuan Chen, Tsungyi Ho, and Bei Yu. Paplace: Performance-driven differentiable analog placement. *ACM Transactions on Design Automation of Electronic Systems*, 31(5):1–23, 2026.

Yuxuan Yin, Yu Wang, Boxun Xu, and Peng Li. Ado-llm: Analog design bayesian optimization with in-context learning of large language models. In *Proceedings of the 43rd IEEE/ACM International Conference on Computer-Aided Design*, pp. 1–9, 2024.

Xi Yu, Dmitrii Torbunov, Soumyajit Mandal, and Yihui Ren. Autosizer: Automatic sizing of analog and mixed-signal circuits via large language model (llm) agents. *arXiv preprint arXiv:2602.02849*, 2026.

Haoyi Zhang, Xiaohan Gao, Haoyang Luo, Jiahao Song, Xiyuan Tang, Junhua Liu, Yibo Lin, Runsheng Wang, and Ru Huang. Sageroute: Synergistic analog routing considering geometric and electrical constraints with manual design compatibility. In *2023 Design, Automation & Test in Europe Conference & Exhibition (DATE)*, pp. 1–6. IEEE, 2023.

Haoyi Zhang, Xiaohan Gao, Zilong Shen, Jiahao Song, Xiaoxu Cheng, Xiyuan Tang, Yibo Lin, Runsheng Wang, and Ru Huang. Sageroute 2.0: Hierarchical analog and mixed signal routing considering versatile routing scenarios. In *2024 Design, Automation & Test in Europe Conference & Exhibition (DATE)*, pp. 1–6. IEEE, 2024.

Haoyi Zhang, Weijian Fan, Xiaohan Gao, Bingyang Liu, Runsheng Wang, and Yibo Lin. Panda: An llm-enhanced performance-driven analog design framework bridging design intent and layout generation. *arXiv preprint arXiv:2606.15052*, 2026a.

Zhishuai Zhang, Xintian Li, Shilong Liu, Aodong Zhang, Lu Jie, and Nan Sun. Ams-io-bench and ams-io-agent: Benchmarking and structured reasoning for analog and mixed-signal integrated circuit input/output design. In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 40, pp. 1579–1586, 2026b.

Andrew Zhao, Daniel Huang, Quentin Xu, Matthieu Lin, Yong-Jin Liu, and Gao Huang. Expel: Llm agents are experiential learners. In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 38, pp. 19632–19642, 2024.

A BENCHMARK TASKS

Table 5 lists the six tasks behind Tables 1, 2 and 4. Tasks 1–5 are OTAs from AMS-SizingBench (Yu et al., 2026): the netlist template, testbench and specification lines are upstream's, and one sizing point per topology was chosen as the feasible candidate whose tightest normalised specification margin is closest to 0.3, so that the layout task is neither trivial nor infeasible. Tasks 2 and 4 reuse the sizing of Tasks 1 and 3 with a smaller load capacitor and a correspondingly higher UGBW target, which turns a task whose unconstrained ALIGN layout is already deliverable into one where it is not (Table 7). Task 6 is the StrongARM comparator of the CABAgent Code-a-Chip notebook (Hu et al., 2026), with upstream's topology and sizing; upstream ships no testbench or specification targets for this design, so the testbench and the three targets are ours. Every device is expressed as m parallel copies of a two-finger unit transistor, which is the form ALIGN's SKY130 primitive generator builds; total width and finger width are unchanged by this rewriting.

Table 5: Six benchmark tasks on SkyWater sky130A. Tasks 1–5 are from AMS-SizingBench, and Task 6 is from the CABAgent Code-a-Chip notebook. A layout is deliverable when it passes DRC and LVS and meets every specification in post-layout simulation. For amplifiers, $\mathrm{FoM} = \text{gain}\,[\mathrm{dB}] \times \mathrm{UGBW}\,[\mathrm{MHz}]/\text{power}\,[\mu\mathrm{W}]$.

Task	Circuit	Devices	Load	Specifications (post-layout)
1	Telescopic OTA	10	1 pF	gain $>$ 18 dB, UGBW $>$ 3 MHz, power $<$ 65 μW, FoM $>$ 0.08
2	Telescopic OTA	10	17 fF	gain $>$ 18 dB, UGBW $>$ 146.95 MHz, power $<$ 65 μW, FoM $>$ 0.08
3	Five-transistor OTA	6	1 pF	gain $>$ 15 dB, UGBW $>$ 10 MHz, power $<$ 65 μW, FoM $>$ 0.5
4	Five-transistor OTA	6	30 fF	gain $>$ 15 dB, UGBW $>$ 216.13 MHz, power $<$ 65 μW, FoM $>$ 0.5
5	Current-mirror OTA	10	1 pF	gain $>$ 15 dB, UGBW $>$ 10 MHz, power $<$ 65 μW, FoM $>$ 0.35
6	StrongARM comparator	15	10 fF	500 MHz clock, 10 mV input at 0.9 V common mode: delay $<$ 1 ns, $V_{\mathrm{out,high}} >$ 1.62 V, $V_{\mathrm{out,low}} <$ 0.18 V

B IMPLEMENTATION DETAILS

B.1 EDA FLOW

Every candidate constraint set is evaluated by the same open-source flow, for every method. Table 6 lists the tools and the versions used for all reported runs. The flow runs the stages below in order and stops at the first failure; a candidate that stops before post-layout simulation has no FoM and is never deliverable.

1. **Layout generation.** ALIGN reads the sized netlist and the constraint JSON, builds primitives, and runs placement and routing. A constraint file that ALIGN's schema rejects, or a constraint set for which the placer finds no solution, returns in one to two seconds with no layout.
2. **DRC.** `magic` checks the GDS against the `sky130A` rule deck. The violation count is reported to the agents.
3. **LVS.** `magic` extracts a netlist from the GDS and `netgen` compares it with the source netlist. Device and pin matches and parameter mismatches are reported verbatim.
4. **Parasitic extraction.** `magic` extracts lumped capacitances (C-only); the total extracted capacitance on each net is reported to the agents in fF.
5. **Stress-geometry annotation.** `klayout` reads the diffusion geometry (SA, SB, SD) of every gate off the GDS and annotates the extracted netlist, so that the BSIM4 length-of-

diffusion model is evaluated on the drawn geometry. A run on which any gate cannot be annotated is scored as a simulation failure.

6. **Post-layout simulation.** `ngspice` runs the task testbench on the annotated extracted netlist at the typical corner and 1.8 V.
7. **Specification check and FoM.** The measured metrics are checked against the task specifications in Table 5 and scored by Equation 3. The area reference A_0 is the area of the unconstrained ALIGN layout of the task (Table 7), which is recorded once per task and never shown to the agents.

Table 6: Tools of the EDA flow. All methods share one flow and one process design kit.

Stage	Tool	Version
Placement and routing	ALIGN (ALIGN-public)	0.9.8
SKY130 adapter	ALIGN-pdk-sky130	with width, unit-cell height, gate length and fin-pitch patches
PDK	SkyWater `sky130A` (via volare)	–
DRC, extraction	magic	8.3.683
LVS	netgen	1.5.323
Stress geometry	klayout (Python module)	0.30.12
Simulation	ngspice	47
LLM	`gpt-5.4`	temperature 0

Table 7: Area reference A_0 of each task: the unconstrained ALIGN layout, measured once and used as the denominator of the area score in Equation 3. Whether that layout is itself deliverable is shown in the last column.

Task	Circuit	A_0 (μm^2)	Unconstrained ALIGN layout
1	Telescopic OTA, 1 pF	521.2	deliverable (4/4 specs)
2	Telescopic OTA, 17 fF	521.2	misses UGBW (3/4 specs)
3	Five-transistor OTA, 1 pF	523.6	deliverable (4/4 specs)
4	Five-transistor OTA, 30 fF	523.6	misses UGBW (3/4 specs)
5	Current-mirror OTA, 1 pF	972.3	fails LVS
6	StrongARM comparator	890.8	deliverable (3/3 specs)

B.2 Constraint Vocabulary

The Constraint Planner does not write ALIGN constraint files directly. It emits *action bundles*, each action naming a family, its targets (device or net names from the netlist), an optional magnitude, and a payload. A deterministic compiler translates a bundle into ALIGN constraint objects and merges them onto the retained constraint set. Table 8 lists the 17 families. The first seven wrap the most common ALIGN constraints with bounded numeric payloads; the next nine expose the remainder of ALIGN's constraint schema, for which the action carries the ALIGN JSON object itself and the Constraint Planner is given a one-line catalogue entry per constraint class; `prune` removes a constraint already in force. BO and ICPI act on ALIGN constraints from the same schema (Appendix B.4).

B.3 Iteration Structure and Hyperparameters

One refinement iteration of MONA proceeds as follows. (1) The Failure Analyst is called on the reports of unsuccessful evaluations: failures in layout generation, DRC, LVS, extraction or simulation, and unmet specifications. (2) Short-term Memory is queried with the current situation, and Long-term Memory returns a fixed-quota bundle (2 run paths, 4 lessons and 3 repair entries) retrieved from earlier tasks of the chain by circuit class. (3) The Constraint Planner is called with full memory and returns a strategy and ranked bundles; the first bundle is the **Exploit** candidate and the highest-ranked bundle whose set of action families differs from it is the **Diversify** candidate. (4) The Constraint Planner is called a second time with Long-term Memory withheld; its first bundle

Table 8: Action families available to the Constraint Planner and the ALIGN constraints they compile to. Magnitudes are integers within the stated bounds; distances are in nanometres; routing layers are M1–M5.

Family	ALIGN constraint(s)	Payload
net_weight	NetPriority	weight $\in [1, 10]$; targets: nets
routing_priority	NetConst (criticality)	criticality $\in [1, 10]$, optional shield net
routing_strength	MultiConnection	multiplier $\in [1, 6]$ (parallel wires)
symmetry	SymmetricBlocks / SymmetricNets	mode $\in$ {blocks, nets}, axis $\in$ {V, H}, device pairs with identical parameters
placement_order	Order	direction $\in$ {horizontal, vertical, left_to_right, top_to_bottom}, abut
grouping	GroupBlocks / DoNotIdentify	mode $\in$ {group, closer, do_not_identify}
generator	GroupBlocks.generator	pattern $\in$ {single_device, cc, id, ratio_devices, ncc}, shared diffusion, body, height
alignment	Align, AlignInOrder, Floorplan	ALIGN JSON
spread	Spread	ALIGN JSON
compaction	CompactPlacement, AspectRatio, Boundary	ALIGN JSON
spacing	BlockDistance, HorizontalDistance, VerticalDistance	ALIGN JSON
boundary_placement	PlaceOnGrid	ALIGN JSON
matching	SameTemplate, GuardRing, GroupCaps	ALIGN JSON
routing_layer	Route, DoNotRoute	ALIGN JSON
ports	PortLocation, PowerPorts, GroundPorts, ClockPorts	ALIGN JSON
compiler	ConfigureCompiler	flags (merge parallel/series devices, remove dummies, fix source/drain, identify array, same template, auto constraint)
prune	(removal)	ids of committed constraints to remove

is the **Explore** candidate. (5) The three candidates are compiled and evaluated by the EDA flow (Appendix B.1). (6) The Verification Gate ranks the iteration's candidates and compares the winner with the retained layout: a candidate that meets every specification replaces a retained layout that does not; among candidates on the same level, the candidate must improve the FoM of Equation 3 by more than 10^{-3}, the measured spread between two routings of the same placement; otherwise the retained layout is kept. A candidate that is not DRC and LVS clean is never committed. (7) On a commit, the winner's constraint set becomes the retained constraint set on which later candidates are built; on a refusal the retained set does not change. (8) The Memory Distiller is called once on the iteration's trajectory and writes one lesson into Short-term Memory. At the end of a run, the run's committed path and repair entries are promoted to Long-term Memory, together with the lessons that were confirmed by a commit; lessons that were repeatedly contradicted are stored as warnings.

Table 9 lists the settings of the reported experiments. The two protocols differ only in the stopping rule: the main experiments and ablations run a fixed number of iterations, while the efficiency analysis runs a fixed number of EDA calls.

Task chains and Long-term Memory. Long-term Memory has content only across tasks. An independent run of the main experiments is therefore one *chain*: the six tasks in the fixed order Task 1, 2, 3, 4, 5, 6 (telescopic OTA at 1 pF and 17 fF, five-transistor OTA at 1 pF and 30 fF, current-mirror OTA, StrongARM comparator), sharing one Long-term Memory bank that starts empty. The first task of every chain therefore runs with an empty Long-term Memory, and within-class transfer

Table 9: Hyperparameters of MONA and the two evaluation protocols.

Setting	Main experiments and ablations (Tables 1, 2, 3)	Efficiency analysis (Table 4)
Iterations per run	5	until the budget of 30 EDA calls is spent
Candidates evaluated per iteration	3 (Exploit, Diversify, Explore)	3
Initial layout	unconstrained ALIGN layout (the area reference A_0)	unconstrained ALIGN layout, evaluated as call 1
Independent runs per task	15	5
Early stopping	off	off
Commit threshold on FoM	10^{-3}	10^{-3}
Long-term Memory retrieval quota	2 paths, 4 lessons, 3 repairs	same
Long-term Memory sharing	one bank per chain of 6 tasks	one bank per chain of 6 tasks
EDA timeout per call	600 s	600 s
LLM	`gpt-5.4`, temperature 0	same

(Task 2 after Task 1, Task 4 after Task 3) is part of every chain. Chains are independent of each other. Each ablation arm runs its own 15 chains under the same protocol.

Ablation arms. *w/o ALS* removes the Topology Analyst, the Failure Analyst and the Constraint Planner; candidates are drawn as uniformly random legal single actions from the vocabulary of Table 8 (three per iteration, seeded per chain). The Verification Gate remains active. *w/o VAM* removes the Verification Gate, the Memory Distiller and both memory tiers; the Constraint Planner sees only its own within-run history. *w/o Gate* removes the Verification Gate only; the Memory Distiller and both memory tiers stay active and are read by the Constraint Planner.

B.4 Baselines

ALIGN. The unconstrained ALIGN layout of each task, evaluated once by the flow of Appendix B.1. It is the initial layout of every method and the area reference A_0 (Table 7); in the efficiency protocol it is also the first EDA call of every ICPI and MONA run.

ICPI. We preserve ICPI's released agent logic, prompts, design journal, retry policy, and control flow, while making only three runtime interface substitutions to connect it to our shared evaluation flow. (1) The synthetic oracle is replaced by the flow of Appendix B.1; ICPI's layout parameters are translated into the ALIGN constraints of Table 8 (net weights to `NetPriority`, routing priorities to `NetConst`, wire widths to `MultiConnection`, placement bias to `Order`, symmetry pairs to `SymmetricBlocks`). (2) ICPI's LLM router is pointed at the same `gpt-5.4` deployment at temperature 0. (3) ICPI's internal FoM is defined over gain, UGB, CMRR and PM and returns 0 when any is missing; since no task here measures CMRR, its formula is kept but evaluated over the metrics each task declares, the same metric set every method is scored on.

BO. Bayesian optimisation with Optuna's tree-structured Parzen estimator over the parameter space ICPI exposes on ALIGN, built from the netlist alone: an integer net weight in $[1, 10]$ and a routing criticality in $[1, 10]$ per signal net, a wire multiplier in $[1, 6]$ per signal net, a placement bias in $[-1, 1]$ per device, and a binary symmetry switch per device pair with identical parameters (the only pairs ALIGN's validator accepts). The objective is the FoM of Equation 3, 0 when no layout or no simulation results. Under the 5-iteration protocol BO makes exactly 5 evaluations with 2 random start-up trials, so that three trials are TPE-guided; under the 30-call protocol it makes 30 evaluations with 5 start-up trials. Seeds are $(r-1) \times 100 + t$ for repeat r and task index t, so that no random stream is shared across tasks.

B.5 Compute

The chains of the main experiments and ablations were run five in parallel, with the six tasks of a chain in series; the LLM is accessed through an API. An EDA call that produces a layout takes 160–260 s through placement, routing, DRC, LVS, extraction and simulation; a call whose constraint set

the placer rejects returns in 1–2 s. A 5-iteration MONA run takes 1 000–2 300 s of EDA time and 60–130 K LLM tokens, a 30-call run 0.5–1.7 h of wall-clock time. The 270 ablation runs took 26 h of wall-clock time.

B.6 Reproduction

The code, the six benchmark tasks (netlists, testbenches, specifications and the recorded area references), the ALIGN adapter patches, the ICPI bridge, the BO runner, the drivers of every experiment, and the raw data of every reported run will be released with the camera-ready version.

C Algorithm and Agent Interfaces

Algorithm 1 gives one run of MONA on one task; the notation is that of Section 3.1. $\mathcal{M}_S$ is Short-term Memory, private to the run, and $\mathcal{M}_L$ is the Long-term Memory bank shared along a chain of tasks. $\varepsilon = 10^{-3}$ is the commit threshold, the measured spread between two routings of one placement, and $\mathrm{clean}(x)$ is 1 when L_x passes DRC and LVS. Every LLM call returns JSON validated against a fixed schema; the four calls and what they see are summarised in Table 10. The full system prompts are released with the code.

Algorithm 1 MONA: one run on task $\mathcal{I} = (G, P, B, S)$ with T iterations

Require: sized netlist G, PDK P, testbench B, specifications S; Long-term Memory $\mathcal{M}_L$; iterations T
Ensure: the retained deliverable layout L^* and its constraint set x^*

$x^* \leftarrow \emptyset$, $L^* \leftarrow$ none, $\mathcal{M}_S \leftarrow \emptyset$
$\mathcal{A} \leftarrow$ TopologyAnalyst(G) ▷ pairs, mirrors, critical nets, circuit class κ
for $t = 1, \ldots, T$ **do**
  $\mathcal{D} \leftarrow$ FailureAnalyst(report, $G, x^*, \mathcal{A}$) **if** the last evaluations were unsuccessful **else** $\emptyset$
  $m_S \leftarrow$ retrieve$(\mathcal{M}_S,$ situation$)$; $m_L \leftarrow$ retrieve$(\mathcal{M}_L, \kappa)$ ▷ 2 paths, 4 lessons, 3 repairs
  (strategy, $\langle b_1, b_2, \ldots \rangle$) $\leftarrow$ ConstraintPlanner$(G,$ report$(L^*), \mathcal{A}, \mathcal{D}, m_S, m_L,$ history$)$
  $x_{\text{exploit}} \leftarrow b_1$; $x_{\text{diversify}} \leftarrow$ first b_k whose action families differ from those of b_1
  $x_{\text{explore}} \leftarrow$ first bundle of ConstraintPlanner$(G,$ report$(L^*), \mathcal{A}, \mathcal{D}, m_S,$ history$)$ ▷ Long-term Memory withheld
  for all $x \in \{x_{\text{exploit}}, x_{\text{diversify}}, x_{\text{explore}}\}$ **do** ▷ three EDA evaluations
    $c_x \leftarrow$ Compile(x, x^*) ▷ ALIGN constraints merged onto the retained set
    $(L_x, \mathrm{clean}(x), D(x), m_x, F_x) \leftarrow$ EDA(G, P, B, S, c_x) ▷ ALIGN, DRC, LVS, PEX, simulation, Eq. 3
  end for
  $w \leftarrow \arg\max_x \big(\mathrm{clean}(x),\ D(x),\ F_x\big)$ in lexicographic order ▷ DRC and LVS first, then specifications, then FoM
  if $\mathrm{clean}(w)$ **and** $\big(L^* =$ none **or** $D(w) > D(x^*)$ **or** $(D(w) = D(x^*)$ **and** $F_w > F_{x^*} + \varepsilon)\big)$ **then** ▷ Verification Gate
    $x^* \leftarrow c_w$, $L^* \leftarrow L_w$ ▷ the retained layout changes only here
  end if
  $\mathcal{M}_S \leftarrow \mathcal{M}_S \cup \{$MemoryDistiller(trajectory$_t$, Δmetrics$)\}$ ▷ one lesson, in roles and mechanisms
end for
$\mathcal{M}_L \leftarrow \mathcal{M}_L \cup$ promote$(\mathcal{M}_S)$ ▷ committed path, repairs, confirmed lessons; contradicted lessons as warnings
return L^*, x^*

D Additional Results

D.1 LLM Token Cost of the Main Experiments

Table 11 gives the LLM token cost of the GPT-5.4 runs of Table 1: thousands of tokens spent per deliverable layout, pooled over the 15 runs of each task. MONA spends 10.4K tokens per deliverable

Table 10: The four LLM agents: when each is called, what it is given, and what it returns. All calls use `gpt-5.4` at temperature 0 and return schema-validated JSON.

Agent	Called	Input	Output
Topology Analyst	once per run	the netlist (every device with model, W, L, nf, m and D/G/S/B nets; every net with fanout, port and supply flags) and the last verification state	matched pairs (same model and size), differential pairs, ratioed same-gate groups, the constraint families likely to move the binding specification, a one-paragraph reading, and a circuit-class key that indexes Long-term Memory
Failure Analyst	on unsuccessful evaluations: failures in layout generation, DRC, LVS, extraction or simulation, and unmet specifications	the stage reached, the tools' own messages, the netlist, the constraints in force, and the Topology Analyst's reading	a failure signature, a one-sentence symptom, a likely cause, and the constraint families that could address it
Constraint Planner	every iteration: once with full memory (Exploit, Diversify) and once with Long-term Memory withheld (Explore)	the netlist with extracted net capacitances, the verification report, the analysis and diagnosis, retrieved Short-term and Long-term Memory, this run's history, and the constraint vocabulary of Table 8	a one-sentence strategy and ranked candidate bundles, best first, each with a rationale and a list of actions
Memory Distiller	every iteration, after the EDA evaluations	the iteration's trajectory (every candidate, the families it touched, the roles of the nets it acted on, whether it laid out, its FoM, its slot) and the signed change in FoM, area and critical-net capacitance	one lesson (title, description, one to three sentences of content) written in roles and mechanisms, never in instance names

layout against 22.2K for ICPI. ICPI's 317.9K on StrongARM is the cost of 15 runs in which only three EDA calls produce a deliverable layout.

Table 11: **LLM token cost of the main experiments with GPT-5.4.** Thousands of LLM tokens (prompt and completion) per deliverable layout, pooled over the 15 runs of each task; Overall pools all 90 runs. The divisor is the number of EDA calls that produced a deliverable layout during the run, including layouts that ICPI later replaces, so a cell can hold a value where the success rate of Table 1 is low. Lower cost per column in bold.

Method	Tele-OTA-1p	Tele-OTA-17f	5T-OTA-1p	5T-OTA-30f	CM-OTA	StrongARM	Overall
ICPI	33.2	36.5	9.4	**13.6**	16.8	317.9	22.2
MONA (Ours)	**5.8**	**11.3**	**8.9**	17.5	**11.5**	**11.1**	**10.4**

D.2 Per-Task Ablation Results

Table 12 breaks the success rates of Table 3 down by task. The three ablation arms run the protocol of the main experiments (15 chains of the six tasks, 5 iterations per run, the same initial layouts, one Long-term Memory bank per chain) and are defined in Appendix B.3. Two facts are visible only

at this granularity. First, the contribution of ALS is concentrated on the tasks whose unconstrained ALIGN layout must be repaired: on Task 5, where it fails LVS, 8 of the 15 w/o ALS runs end with no deliverable layout after the fifth iteration. Second, without the Verification Gate the loss is concentrated on the tasks whose specifications are tight: on Task 4, whose UGBW specification is tight, 1 of 15 runs holds a deliverable layout after the fifth iteration although 14 of 15 saw one, because the last candidate replaces the retained layout whether or not it is deliverable.

Table 12: Per-task success of the ablation arms: successful runs out of 15 per task (15 chains, 5 iterations per run). The last row gives the totals behind the success rates of Table 3.

Task	MONA	w/o ALS	w/o VAM	w/o Gate
1: Tele-OTA-1p	15/15	15/15	12/15	11/15
2: Tele-OTA-17f	15/15	14/15	10/15	9/15
3: 5T-OTA-1p	15/15	15/15	10/15	10/15
4: 5T-OTA-30f	14/15	14/15	5/15	1/15
5: CM-OTA	15/15	7/15	7/15	6/15
6: StrongARM	15/15	15/15	11/15	6/15
All 90 runs	89/90	80/90	55/90	43/90

D.3 Candidate Slots

Table 13 breaks Figure 4 down by task. On Task 1, the first task of every chain, the Long-term Memory bank is empty, and on Task 6, the only comparator, it holds no comparator experience; on these two tasks the Explore candidate is counted under Exploit. On Tasks 2–5, where Long-term Memory holds experience from earlier tasks of the chain, the three candidates deliver the final layout about equally often; the Explore candidate, proposed without Long-term Memory, is retained as often as the two proposed with it. Figure 7 shows the same breakdown as a share of the delivered layouts of each task (panel a) and, on Tasks 2–5, follows every generated candidate through the run (panel b). Of the 299 candidates generated in each slot, 68–71% yield a layout that passes DRC and LVS, 12–14% are accepted by the Verification Gate, and 6–8% are the layout retained at the end of the run; at every stage the three slots differ by less than the width of their 95% Wilson intervals.

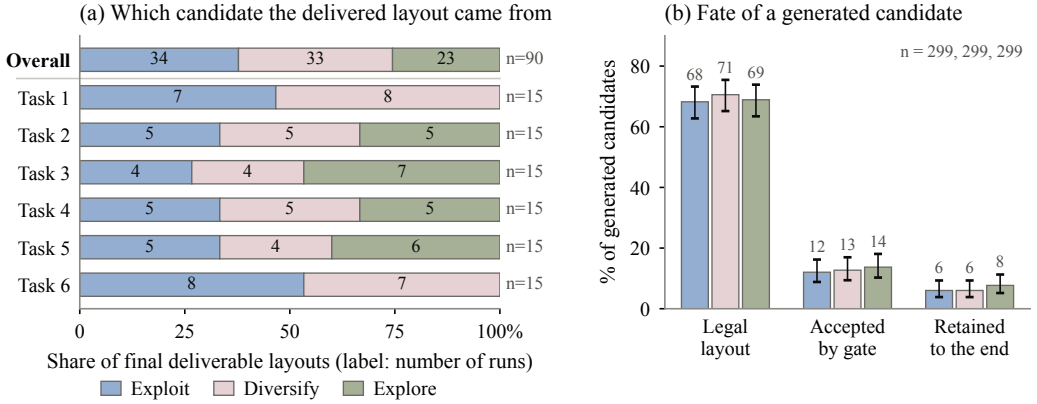


Figure 7: Contribution of the three candidate slots in the main experiments (90 runs of MONA). (a) Share of the final deliverable layouts delivered by the Exploit, Diversify, and Explore candidate, per task and overall; bar labels are numbers of runs, and on Tasks 1 and 6 the Explore candidate is counted under Exploit as in Table 13. (b) Fate of a generated candidate on Tasks 2–5, where all three candidates are generated in every iteration ($n = 299$ per slot): the fraction that passes DRC and LVS, that is accepted by the Verification Gate, and that is the layout retained at the end of the run. Error bars are 95% Wilson intervals.

Table 13: Candidate slot of the layout delivered by each of the 90 runs of MONA in the main experiments. On Tasks 1 and 6 the Explore candidate is counted under Exploit.

Task	Exploit	Diversify	Explore
1: Tele-OTA-1p	7	8	–
2: Tele-OTA-17f	5	5	5
3: 5T-OTA-1p	4	4	7
4: 5T-OTA-30f	5	5	5
5: CM-OTA	5	4	6
6: StrongARM	8	7	–
All 90 runs	34	33	23

E CASE STUDY

We walk through one run of the main experiments, iteration by iteration: run 11 of MONA on Task 6, the StrongARM comparator, which is the last task of its chain. The run makes 15 EDA calls in 5 iterations, 8 of which produce deliverable layouts, and spends 73K LLM tokens and 2 490 s of EDA time. Task 6 is the only comparator of the chain, so Long-term Memory holds no comparator experience. Table 14 lists every candidate; Figure 8 shows the unconstrained ALIGN layout of the task and the delivered layout.

What the gate did. Three of the five iterations end in a commit and two in a refusal, and the retained FoM never falls: 0.960, 0.960, 1.043, 1.179, 1.179. Both refusals are of clean, specification-meeting layouts whose FoM is below the retained one by 0.006 and 0.009, that is, above the 10^{-3} routing-noise threshold and in the wrong direction; under a cumulative state either would have replaced a better layout. Iteration 3 shows the specification-first rule: the Exploit candidate prunes six committed constraints and is DRC and LVS clean, but the post-layout testbench reports no delay, so it is not deliverable and cannot be ranked above the two candidates that meet every specification; the iteration's winner is the empty constraint file, which reproduces the unconstrained ALIGN layout at FoM 1.043. Iteration 4 then finds the delivered layout with four pairwise alignment constraints on top of that empty base.

What the memory recorded. The Memory Distiller writes one lesson per iteration in roles and mechanisms rather than instance names. After iteration 1 it wrote: *"In a no-layout start, gentle guidance on a clock port together with higher weighting and routing priority on the critical internal shared nets and outputs can give the placer enough direction to find a clean legal layout. Broad symmetry, grouping, or matching constraints are easier to over-constrain at this stage and should be added only after placement is already robust."* After iteration 4: *"If global matching/alignment constraints are too strong to place, reduce them to local pairwise alignment on matched devices along the signal path. This preserves placer freedom while encouraging a more compact arrangement, which can lower area enough to outweigh small electrical regressions in the overall score."* The refused iterations also leave lessons, for example after iteration 3: *"If a step aimed at area improves area clearly but lowers total score, treat it as right direction, wrong price: keep the guidance that is protecting the sensitive nets and make the compaction move gentler."* Short-term Memory also records each failed candidate with its measured consequence. At the end of the run, the committed path and the lessons that had held were promoted to the chain's Long-term Memory bank; the lessons that never held were dropped. Because Task 6 is the last task of the chain, this run's entries are not read by any later run.

Cost. The run's 73K LLM tokens include the single Topology Analyst call of the first iteration. EDA seconds per iteration are 181, 1 023, 550, 415 and 322; the second iteration's cost is the 600 s ALIGN timeout of its Diversify candidate.

Table 14: Every candidate of run 11 of MONA on Task 6 (StrongARM comparator). *Cand.*: candidate slot. *Outcome*: where the EDA flow stopped, or the FoM and area of a clean layout that meets all three specifications. *Gate*: the Verification Gate's decision on the iteration's winner; the retained layout after the iteration is shown in the last column.

It.	Cand.	Proposed constraints	Outcome	Gate	Retained
1	Exploit	Vertical symmetry over 14 signal-path devices, symmetric output and internal nets, clock ports, compiler flags	no placement (1 s)	–	0.960, 970 μm^2
1	Diversify	`NetPriority` 8 on the two internal nets and 7 on the outputs, `NetConst` criticality 8 on the outputs, `ClockPorts`	clean, 3/3 specs, FoM 0.960, 970 μm^2 (179 s)	accepted (first layout)	
1	Explore	`SameTemplate` on device pairs, grouping, compiler flags	refused by ALIGN's validator (1 s)	–	
2	Exploit	Iteration-1 winner with weights raised to 9/8/9 and `MultiConnection` 2 on the outputs	clean, 3/3, FoM 0.954, 973 μm^2 (164 s)	rejected: -0.006 below the retained FoM	0.960, 970 μm^2
2	Diversify	Routing-layer bounds on the outputs, port sides	ALIGN timeout (600 s)	–	
2	Explore	Two compaction constraints	clean, 3/3, FoM 0.944, 970 μm^2 (257 s)	–	
3	Exploit	Prune 6 committed constraints, net weights 6/5, criticality 6, compaction	DRC and LVS clean, but the testbench reports no delay: specification failure, 1 516 μm^2 (243 s)	–	1.043, 891 μm^2
3	Diversify	Two compaction constraints	clean, 3/3, FoM 0.614, 1 516 μm^2 (143 s)	–	
3	Explore	Prune all 7 net constraints (empty constraint file)	clean, 3/3, FoM 1.043, 891 μm^2 (164 s)	accepted	
4	Exploit	Alignment of 8 signal-path devices	refused by ALIGN's validator (1 s)	–	**1.179, 649 μm^2**
4	Diversify	Two compaction constraints	4 DRC violations, 1 516 μm^2 (212 s)	–	
4	Explore	Four pairwise `Align` constraints on matched devices (bottom edges of two pairs, top edges of two pairs)	clean, 3/3, **FoM 1.179, 649 μm^2** (202 s)	accepted (final)	
5	Exploit	Alignment of two device triples on the signal path	clean, 3/3, FoM 1.010, 922 μm^2 (126 s)	–	1.179, 649 μm^2
5	Diversify	Symmetric nets on the internal and output nets	no placement (1 s)	–	
5	Explore	`NetPriority` 3 and criticality 4 on the outputs, on top of the iteration-4 base	clean, 3/3, FoM 1.170, 673 μm^2 (195 s)	rejected: -0.009 below the retained FoM	

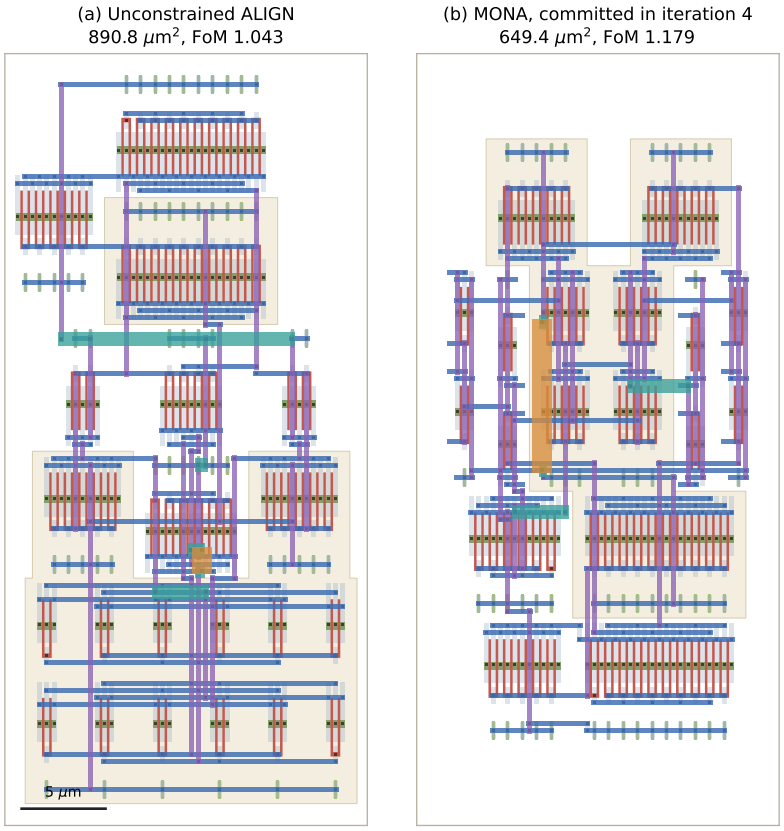


Figure 8: Task 6, StrongARM comparator, drawn at one scale. (a) The unconstrained ALIGN layout, which is the area reference of the task. (b) The layout delivered by run 11 of MONA, committed in iteration 4 with four pairwise alignment constraints and nothing else.