

Retrieve, Schedule, Reflect: LLM Agents for Chip QoR Optimization

Yikang Ouyang, Yang Luo, Dongsheng Zuo, Yuzhe Ma*
The Hong Kong University of Science and Technology (Guangzhou), Guangzhou, China

Abstract

Quality-of-results (QoR) optimization through Engineering Change Orders (ECOs), measured by timing, power, and area, is key to chip signoff. Although commercial EDA tools provide mature optimization algorithms such as gate sizing, threshold-voltage assignment, and buffer insertion, achieving high QoR still heavily depends on iterative human expert intervention. To address this issue and automate chip ECO, we propose an agentic LLM framework that schedules chip optimizations through direct interaction with EDA tools. The agents are grounded in natural language expertise expressed as a search tree through retrieval-augmented generation (RAG). We further improve scheduling quality with Pareto-driven QoR feedback through language reflection. Across eight designs, our framework achieves over 10% better TNS and 25% better WNS than reinforcement learning-based black-box optimization methods. It also achieves up to 4.5% lower area and 1.5% lower power than RL baselines, while running more than 4× faster. Finally, the agent supports customized tasks expressed in natural language, enabling preferential QoR trade-offs. Our work is open-sourced at <https://github.com/YiKangOY/LLM-PO>.

ACM Reference Format:

Yikang Ouyang, Yang Luo, Dongsheng Zuo, Yuzhe Ma. 2026. Retrieve, Schedule, Reflect: LLM Agents for Chip QoR 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/10.1145/3831599.3840323>

1 Introduction

Across modern chip-design flows, ECOs are performed repeatedly from synthesis through signoff to optimize power, performance, and area (QoR). Commonly used algorithms for improving QoR include gate sizing, threshold-voltage (V_{th}) assignment, and buffer insertion/removal [1, 2, 3].

While these algorithms are integrated into EDA tools and can be invoked easily, achieving competitive QoR still requires extensive human effort. Due to the NP-hard nature of chip optimization [4], QoR improvement often saturates and may be trapped in local minima after several iterations [5]. As a result, the optimization becomes an iterative process that requires the intervention of expert engineers. As shown in Figure 1, engineers inspect reports

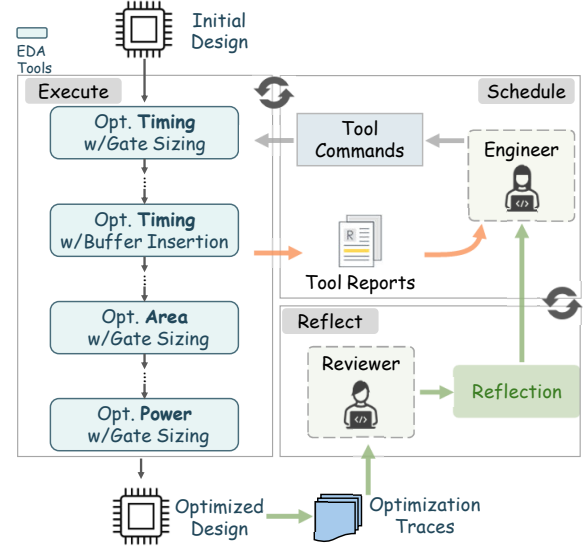


Figure 1: Post-routing ECO optimization as an iterative report-driven schedule-execute-reflect workflow.

from EDA tools and previous ECO traces to select the ECO objective and algorithm, which we call **scheduling**, then invoke EDA tools for execution. When QoR improvement stagnates, engineers refine the optimization strategy and restart new rounds from the initial design. Consequently, this **schedule-execute-reflect** loop is labor-intensive, time-consuming, and demands continuous human monitoring.

Prior works often treat QoR optimization as black-box search over a fixed sequence of EDA tool commands. These efforts predominantly rely on reinforcement learning since it can optimize over black-box objectives with only numerical QoR feedback [6, 7, 8, 2]. Despite enabling a certain level of automation, these methods face limitations because they primarily rely on numerical QoR feedback. They cannot comprehend the rich information expressed in natural-language tool reports and logs as human engineers do, leaving critical context for ECO unused and often leading to suboptimal QoR. Moreover, supporting customized, open-ended optimization tasks remains difficult, since they rely on predefined reward functions [9].

To address these challenges and enable efficient, high-QoR, and customizable chip optimization, we propose the first agentic large language model (LLM) framework that converts natural-language ECO preferences into iterative EDA tool interactions. The framework operates in a **retrieve-schedule-reflect** loop using multiple agents. First, it **retrieves** relevant optimization expertise from a library formulated as a search tree, where paths encode human-interpretable ECO strategies described in natural language. This grounding allows the agent to understand report states and user preferences with expert domain context rather than relying solely

*Corresponding author.



on numerical QoR feedback. Based on the retrieved expertise, the agent then **schedules** the next objective, strategy, and legal EDA tool options. Furthermore, to continuously refine decisions under multi-objective QoR trade-offs, we introduce a Pareto-driven **reflection** mechanism. The agent identifies Pareto-optimal optimization traces, summarizes effective preference-aware scheduling strategies, and incorporates these reflections to improve future schedules.

Our contributions are summarized as follows.

- (1) We introduce **Retrieve–Schedule–Reflect**, an LLM multi-agent framework for preference-driven ECO optimization through iterative scheduling of EDA tool usage.
- (2) We organize expert ECO scheduling knowledge as a natural-language search tree for RAG, grounding scheduling decisions in expert strategies and tool manuals through RAG.
- (3) We propose Pareto-driven reflection to improve ECO quality by learning from historical timing–power–area trade-offs.
- (4) On eight designs placed and routed with ASAP7 PDK [10], our framework achieves over 10% better TNS and 25% better WNS than RL-based methods. It also achieves up to 4.5% lower area and 1.5% lower power than them, and delivers higher hypervolume with more than 4× speedup.
- (5) Our framework supports customized ECO preferences specified in natural language, enabling controllable trade-offs without task-specific reward engineering.

2 Preliminaries

2.1 ECO and LLM for Chip Design

Common ECO algorithms include gate sizing [4, 11], threshold-voltage assignment, and buffer insertion [12, 13]. These algorithms have been integrated into EDA tools [14, 15], but achieving satisfactory QoR still requires engineers to determine objective priorities, algorithm choices, and tool options from reports [16].

LLMs have recently been used in chip design as tool-using agents that read reports/logs and propose edits to flow scripts. For example, ChipNeMo studies domain adaptation for script generation and bug analysis [17], ChatEDA generates scripts to automate RTL-to-GDSII flow [18], and ORFS-agent tunes RTL-to-GDS flow parameters with an LLM [19]. Different from these works, our framework uses LLMs to understand user-specified QoR preferences and schedule multi-iteration ECO with EDA tool.

2.2 LLM for Task Scheduling

Many LLM agents act as task schedulers by interpreting natural-language goals, choosing tools, observing intermediate results, and updating their next decisions. ReAct is a representative reasoning-action formulation that interleaves explicit LLM reasoning with environment actions, allowing the next-step plan to be updated after each observation [20].

Two mechanisms are especially useful for improving scheduling quality in iterative preference optimization. First, **retrieval-augmented generation (RAG)** lets the agent query external knowledge to guide scheduling with documentation and expert examples [21]. Second, **reflection** reviews summaries of failures/successes and uses them to refine future schedules, improving behavior in iterative workflows [22].

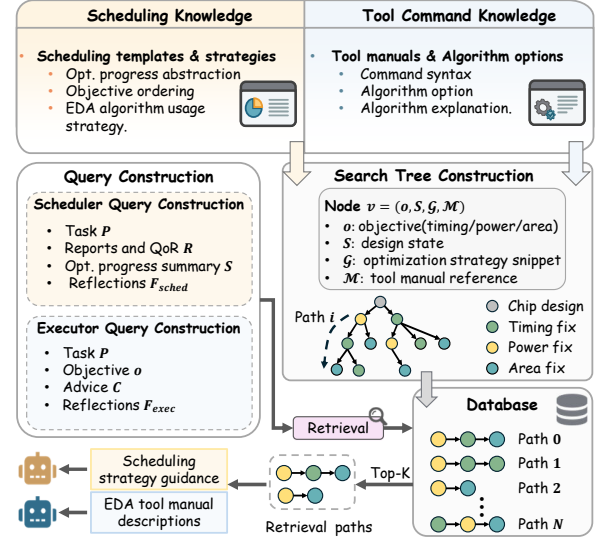


Figure 2: Scheduling strategy constructed as a search tree and path-based RAG.

2.3 Problem Formulation

We formulate post-routing ECO as a scheduling and execution process of EDA tool usage under a user-defined natural-language optimization task. Let $D^{(0)}$ denote the routed design with report $R^{(0)}$ and $\mathcal{P}$ denote the designer-specified task. At iteration i , the scheduler observes $\mathcal{P}$ and accumulated reports $R^{(0:i-1)}$, then selects an optimization objective

$$o^{(i)} \in O = \{\text{timing, power, area}\}, \quad (1)$$

and generates strategy guidance for the executor. The executor converts this guidance into a legal ECO action $A^{(i)}$ with tool-specific options. The EDA tool applies the action and returns an updated design and report:

$$(D^{(i)}, R^{(i)}) = \mathcal{E}(D^{(i-1)}, A^{(i)}). \quad (2)$$

For a completed run with n iterations of ECO, the framework extracts a post-optimization QoR tuple

$$\mathbf{z} = (z_{\text{timing}}, z_{\text{power}}, z_{\text{area}}) \quad (3)$$

from $R^{(n)}$. As optimization iterations accumulate, QoR improvements tend to saturate [5]. Thus we restart the optimization for multiple rounds, each producing a candidate optimized design. The final output is the candidate design that best satisfies the natural-language preference $\mathcal{P}$.

3 Search Tree-based Retrieval-augmented Scheduling

Effective ECO scheduling requires reusable expert knowledge about objective trade-offs and legal execution compliant to EDA tool manuals. Retrieval-augmented Generation (RAG) provides a natural mechanism to supply this knowledge at runtime, allowing the agents to condition each decision on relevant optimization strategies and legal tool options. Inspired by decision-tree formulations for sequential chip optimization [23], we model expert schedules as a search tree (ST) and retrieve relevant paths at runtime.

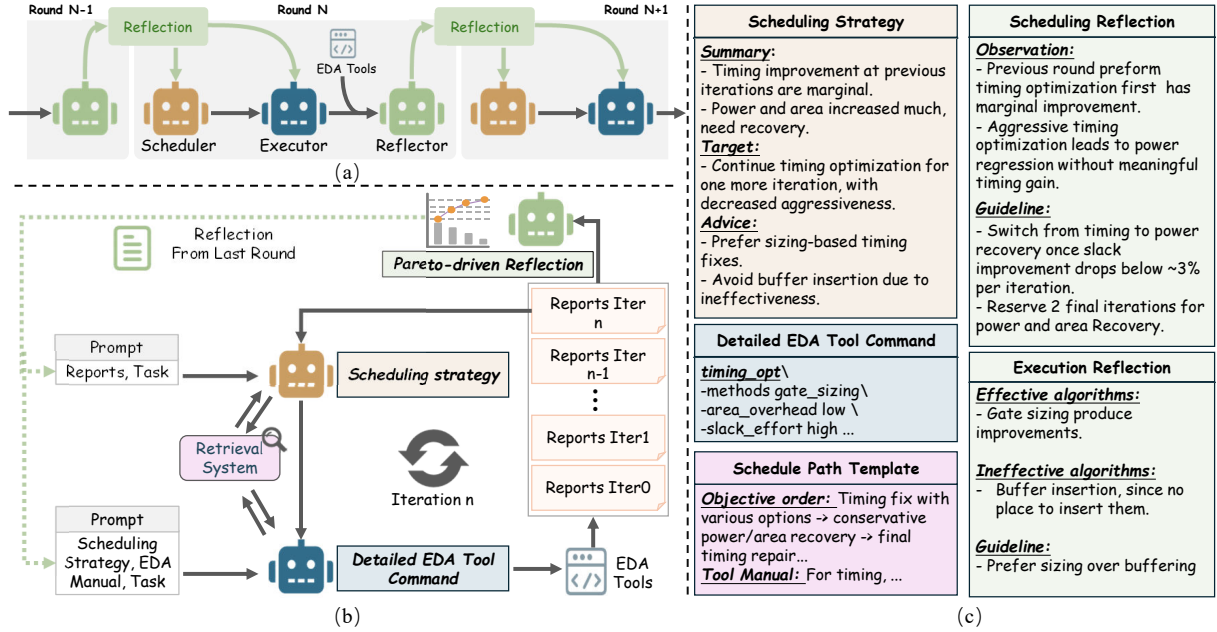


Figure 3: Overview of our framework. (a) Rounds of chip optimization scheduling. (b) Intra-round retrieval, scheduling, and execution. (c) Example prompt and agent response snippets.

3.1 Search Tree Representation and Knowledge Storage

The ST is built from papers, reports, expert ECO experiences, and tool manuals, as illustrated in Figure 2. Each node is associated with an objective $o \in \mathcal{O} = \{\text{timing, power, area}\}$, a natural-language design state s , a strategy snippet $\mathcal{G}$ describing how to improve o , and objective-specific manual snippets $\mathcal{M}_o$: Formally, we define a node as

$$v \triangleq (o, s, \mathcal{G}, \mathcal{M}_o), \quad o \in \mathcal{O}. \quad (4)$$

An ST path is a variable-length expert scheduling trajectory:

$$p = (v_1, v_2, \dots, v_L), \quad L \text{ arbitrary}. \quad (5)$$

Paths have no fixed length because satisfying a user preference may require multiple EDA tool calls and objective switches. One path can represent a complete expert scheduling process and its associated QoR evolution. It also captures the strategy that designers use to select subsequent objectives and the corresponding algorithms to invoke in EDA tools. We extract all top-to-leaf paths into a database $\mathcal{D}$ for retrieval.

3.2 Path-based RAG

Given a natural-language query x and a candidate ST path p , we compute dense embeddings $e_x = \phi(x)$ and $e_p = \phi(p)$ using the same text encoder ϕ . We measure relevance using cosine similarity as:

$$\text{Sim}(x, u) = \frac{e_x \cdot e_u}{\|e_x\| \|e_u\|}. \quad (6)$$

We then retrieve the most relevant expert guidance via top- k similarity search:

$$\text{TopK}(x, \mathcal{D}) = \arg \max_{p \in \mathcal{D}} \text{top-}k \text{ Sim}(x, p), \quad (7)$$

where $\mathcal{D}$ is the ST path database. The retrieved paths provide the agent with end-to-end scheduling guidance, which the agent uses to make guided decisions and invoke EDA tools correctly.

4 Agentic Schedule–Execute–Reflect Framework

Building upon the ST-based RAG in Section 3, we present an agentic framework for iterative preference-driven QoR optimization as an ECO process. Starting from a routed design, it performs iterative schedule–execute steps within each round. During these iterations, RAG provides expert guidance and tool manuals to improve scheduling quality and ensure the executed algorithms are legal for EDA tools. Once QoR improvement saturates within a round, the framework restarts from the routed design and launches a new round, as shown in Figure 3 (a). Across rounds, the reflector agent summarizes feedback from historical traces and transfers it to the scheduler and executor to improve scheduling quality in subsequent rounds.

4.1 Intra-round Scheduling and Execution

We first describe the intra-round scheduling and execution iterations shown in Figure 3(b). At iteration i of round r , the scheduler interprets the natural-language task, selects the next optimization objective, and forms a high-level strategy. It observes the accumulated EDA reports and QoR history $R_r^{(0:i-1)}$, the user-defined optimization task $\mathcal{P}$, and the previous-round scheduling reflection $\mathcal{F}_{\text{sched}}^{(r-1)}$.

To ground the scheduling in guidance from the ST database $\mathcal{D}$, the scheduler summarizes the user preference, report history, and current QoR state as $s^{(i)}$ and forms a retrieval query:

$$x^{(i)} = \psi_{\text{sched}}(\mathcal{P}, R_r^{(0:i-1)}, s^{(i)}, i, \mathcal{F}_{\text{sched}}^{(r-1)}). \quad (8)$$

Then the scheduler uses $x^{(i)}$ to retrieve the most relevant expert scheduling paths:

$$\hat{\mathcal{Y}}^{(i)} = \text{TopK}(x^{(i)}, \mathcal{D}). \quad (9)$$

Algorithm 1 Iterative ECO scheduling workflow

Require: Optimization task $\mathcal{P}$; ST path database $\mathcal{D}$; manuals $\{\mathcal{M}_o\}_{o \in \mathcal{O}}$

- 1: Initialize reflections $\mathcal{F}_{\text{sched}}^{(0)}$, $\mathcal{F}_{\text{exec}}^{(0)}$ and trace set $\mathcal{S} \leftarrow \emptyset$
- 2: **for** $r = 1$ to R_{max} **do**
- 3: Reset to the initial unoptimized design and obtain initial report $R_r^{(0)}$
- 4: **for** $i = 1$ to I_{max} **do**
- 5: **Scheduling:** summarize state $s^{(i)}$ and form query $x^{(i)}$ (Equation (8))
- 6: Retrieve ST paths; select objective $o^{(i)}$ and strategy guidance $C^{(i)}$
- 7: **Execution:** form manual query $q^{(i)}$ and retrieve tool references (Equation (10))
- 8: Generate and execute command $A^{(i)}$ to obtain report $R_r^{(i)}$ (Equation (12))
- 9: **end for**
- 10: Record trace S_r and terminal QoR vector $\mathbf{z}_r$; update $\mathcal{S} \leftarrow \mathcal{S} \cup \{S_r\}$
- 11: Filter Pareto-optimal traces from $\mathcal{S}$ and update Pareto-guided reflections (Section 4.2)
- 12: **end for**
- 13: **return** optimized design yielded from all rounds of traces under preference $\mathcal{P}$

The retrieved paths $\widehat{\mathcal{Y}}^{(i)}$ provide guidance on objective selection and tool-use strategies that match the current design state and requested preference. Conditioned on $\widehat{\mathcal{Y}}^{(i)}$ and $\mathcal{F}_{\text{sched}}^{(r-1)}$, the scheduler selects the next optimization objective $o^{(i)} \in \mathcal{O}$ and generates strategy guidance $C^{(i)}$.

The executor translates the high-level strategy guidance into an EDA command given $(\mathcal{P}, o^{(i)}, C^{(i)})$. To ensure command legality and objective consistency, it queries the tool manual:

$$q^{(i)} = \psi_{\text{exec}}(\mathcal{P}, o^{(i)}, C^{(i)}), \quad (10)$$

and retrieves objective-oriented references:

$$\widehat{\mathcal{M}}_{o^{(i)}}^{(i)} = \text{TopK}(q^{(i)}, \mathcal{M}_{o^{(i)}}). \quad (11)$$

Conditioned on the retrieved manual $\widehat{\mathcal{M}}_{o^{(i)}}^{(i)}$ and execution reflection $\mathcal{F}_{\text{exec}}^{(r-1)}$, the executor generates an executable command $A^{(i)}$ that implements $C^{(i)}$ while following the EDA manual and prior legal tool-use syntax [24].

Executing $A^{(i)}$ in the EDA tool yields updated design and corresponding reports

$$R_r^{(i)} = \mathcal{E}(R_r^{(i-1)}, A^{(i)}), \quad (12)$$

The updated report is recorded and becomes input to iteration $i + 1$. This intra-round scheduling and execution procedure continues until QoR improvement saturates or the iteration budget is reached.

4.2 Inter-round Pareto-guided Reflection

As more optimization operators are applied, QoR improvement often saturates and gets stuck in a local optimum [16, 5]. To further optimize QoR, our framework draws feedback from previous-round optimization traces and restarts from the initial design with improved scheduling strategy through reflection. Pareto dominance provides a multi-objective perspective to evaluate the trade-offs

among timing, power, and area across rounds [25, 26]. Thus we maintain a set of Pareto-optimal traces and prompt the reflector to generate feedback for the scheduler and executor in the next round from this set of optimal traces, as shown in Figure 3.

4.2.1 Filtering Pareto-optimal Traces Let $\mathcal{Z} = \{\mathbf{z}_1, \mathbf{z}_2, \dots, \mathbf{z}_r\}$ denote the optimized QoR tuple from all completed rounds up to current round r :

$$\mathbf{z}_j \triangleq (z_{j,\text{timing}}, z_{j,\text{power}}, z_{j,\text{area}}) \quad (13)$$

where $\mathbf{z}_j$ is extracted from the report of the optimized design at round j (i.e., from $R_j^{(n_j)}$). Larger $z_{j,\text{timing}}$ indicates better timing, while smaller $z_{j,\text{power}}$ and $z_{j,\text{area}}$ indicate better power and area.

Each QoR tuple corresponds to a complete optimization round trace S_j that records executed commands and resulting reports:

$$S_j \triangleq ((A_j^{(1)}, R_j^{(1)}), (A_j^{(2)}, R_j^{(2)}), \dots, (A_j^{(n_j)}, R_j^{(n_j)})). \quad (14)$$

We compute the Pareto front $\mathcal{Z}_{\text{Pareto}} \subseteq \mathcal{Z}$ under the objectives of maximizing timing and minimizing power and area, then obtain complete optimization traces from the filtered QoR tuples:

$$\mathcal{S}_{\text{Pareto}} = \{S_j \mid \mathbf{z}_j \in \mathcal{Z}_{\text{Pareto}}\}. \quad (15)$$

4.2.2 Scheduler and Executor Reflection Optimal QoR is achieved through synergistic work between schedulers and executors. To benefit both of them through reflection, we prompt the reflector to generate two complementary reflections: a scheduling reflection $\mathcal{F}_{\text{sched}}^{(r)}$ and an execution reflection $\mathcal{F}_{\text{exec}}^{(r)}$.

The scheduler reflection $\mathcal{F}_{\text{sched}}^{(r)}$ extracts patterns of QoR priorities and trade-offs among iterations that lead to Pareto-optimal traces. It guides when to focus on one objective versus when to pursue others, so that the schedule can satisfy the requested preference. The executor reflection $\mathcal{F}_{\text{exec}}^{(r)}$ summarizes which EDA commands and option settings are effective, while identifying ineffective choices to avoid. These reflections are provided to the scheduler and executor in the next round, progressively improving ECO scheduling quality in terms of QoR across rounds.

5 Experimental Results

5.1 Experimental Setup

To evaluate the generality of our approach, we use open-source chip designs implemented in the ASAP7 PDK [10] from the ICCAD'24 dataset [27]. We select 8 designs from this benchmark. We evaluate three core QoR metrics: timing (both WNS and TNS), power, and area. All optimizations and report generation are performed using *Synopsys PrimeECO*, a widely used signoff tool. All experiments are conducted on a machine with two AMD EPYC 7543 32-core CPUs and an NVIDIA RTX 3090 GPU. Unless otherwise specified, we use **GPT-5-mini** (2025-08-07) as the default backbone LLM.

To the best of our knowledge, there is no established baseline that directly targets iterative ECO. We therefore adapt two reinforcement learning (RL) baselines with A3C for architecture search [26] and a PPO algorithm for EDA tool parameter search [28], and train them for 50 episodes (rounds), each with 10 iterations, to search fixed-length sequences of EDA commands from QoR feedback. In addition, we include a human baseline by asking an experienced engineer

Table 1: Chip design statistics and optimization results. Here, “TS” and “WS” denote total slack and worst slack, respectively. Ratio (%) indicates the percentage of the post-ECO metric compared with the original value. The best results are bolded.

Metric	Method	NVDLA partition_m	NVDLA partition_p	ariane136	mempool tile_wrap	aes_256	hidden1	hidden2	hidden5	Avg.	Ratio↓ (%)
TS↑ (ps)	Original	-1.909e+04	-2.073e+05	-2.189e+05	-2.133e+06	-5.774e+04	-4.237e+04	-1.325e+05	-1.512e+05	-3.702e+05	-
	Human	-2.901e+03	-1.174e+04	0.000e+00	0.000e+00	0.000e+00	0.000e+00	0.000e+00	-4.327e+02	-1.884e+03	0.51%
	RL (A3C)	-8.094e+03	-1.480e+05	-1.579e+05	0.000e+00	-2.979e+03	-5.374e+03	-3.587e+02	-6.237e+02	-4.042e+04	10.92%
	RL (PPO)	-1.611e+03	-5.022e+03	0.000e+00	-1.558e+06	0.000e+00	-2.547e+03	-1.010e+03	-4.289e+02	-1.961e+05	52.96%
	Ours	-1.588e+03	-4.739e+03	0.000e+00	0.000e+00	0.000e+00	0.000e+00	0.000e+00	-4.259e+02	-8.441e+02	0.23%
WS↑ (ps)	Original	-3.657e+02	-2.784e+02	-1.667e+02	-4.777e+02	-5.820e+01	-1.314e+02	-1.823e+02	-8.691e+01	-2.184e+02	-
	Human	-9.276e+01	-6.650e+01	3.431e-01	4.700e-01	2.400e+00	8.670e+01	2.330e-01	-3.117e+01	-1.254e+01	5.74%
	RL (A3C)	-2.144e+02	-2.601e+02	-1.935e+02	2.761e+00	-3.310e+01	-1.225e+02	-1.573e+01	-3.117e+01	-1.085e+02	49.68%
	RL (PPO)	-6.468e+01	-4.759e+01	2.182e-01	-3.101e+02	2.400e-03	-8.904e+01	-2.912e+01	-3.117e+01	-7.144e+01	32.71%
	Ours	-6.468e+01	-4.743e+01	1.660e-01	3.700e+01	8.000e-04	3.400e-03	3.960e-02	-3.117e+01	-1.326e+01	6.07%
Area↓ (μm^2)	Original	2.728e+03	9.570e+03	3.266e+04	2.813e+04	2.728e+04	5.355e+03	3.307e+04	2.734e+04	2.077e+04	-
	Human	2.879e+03	9.655e+03	2.938e+04	2.728e+04	2.730e+04	5.362e+03	2.968e+04	2.753e+04	1.988e+04	95.71%
	RL (A3C)	2.757e+03	9.626e+03	3.253e+04	2.831e+04	2.725e+04	5.353e+03	3.307e+04	2.712e+04	2.075e+04	99.90%
	RL (PPO)	2.883e+03	9.653e+03	3.267e+04	2.720e+04	2.731e+04	5.370e+03	3.296e+04	2.744e+04	2.068e+04	99.61%
	Ours	2.898e+03	9.665e+03	2.937e+04	2.735e+04	2.692e+04	5.342e+03	2.947e+04	2.753e+04	1.982e+04	95.43%
Power↓ (W)	Original	5.250e-03	5.450e-02	5.928e-01	4.640e-02	3.037e-01	2.200e-02	5.863e-01	4.440e-01	2.569e-01	-
	Human	5.710e-03	5.680e-02	5.841e-01	4.770e-02	3.056e-01	2.210e-02	5.752e-01	4.544e-01	2.565e-01	99.84%
	RL (A3C)	5.518e-03	5.580e-02	5.928e-01	4.900e-02	3.110e-01	2.200e-02	5.864e-01	4.550e-01	2.597e-01	101.09%
	RL (PPO)	5.866e-03	5.700e-02	5.929e-01	4.790e-02	3.056e-01	2.220e-02	5.861e-01	4.553e-01	2.591e-01	100.87%
	Ours	5.875e-03	5.700e-02	5.842e-01	4.700e-02	3.006e-01	2.200e-02	5.747e-01	4.557e-01	2.559e-01	99.61%

to operate EDA tools for 10 rounds, each with 10 iterations. Our framework runs with the same configuration as the human does.

5.2 Optimization Results

As a common practice, timing slack is treated as a hard constraint and is prioritized first, since negative slack may lead to functional failure of the design. Once timing violations are resolved, power and area are optimized as secondary objectives under the satisfied timing constraint. Our primary optimization preference is thus specified in natural language: **achieve non-negative worst slack (WNS), and once this timing target is reached, minimize the normalized area–power cost, computed as $\text{Area}/\text{Area}_{\text{init}} + \text{Power}/\text{Power}_{\text{init}}$** . This preference is also used to select the optimized design to present from all launched optimization rounds (or episodes for RL baselines). Table 1 compares our method with RL baselines and manual scheduling by an experienced engineer.

Our framework achieves the strongest average timing optimization. For TNS, it improves the original timing violation by 99.77%, compared with 89.08%/47.04% for A3C [26]/PPO [28], which corresponds to 10.7%/52.7% better TNS than A3C/PPO. For WNS, it reduces the average violation from -218.4 ps to -13.26 ps, which corresponds to 43.6%/26.6% better WNS than A3C/PPO, while remaining comparable to the human baseline (-12.54 ps). On designs with remaining negative slack after optimization, such as NVDLA_partition_m/p and hidden5, our method still achieves the least negative slack.

In addition to optimizing timing, our framework also preserves power and area. It achieves the lowest average area among all optimized results (95.43% of the original design) and the lowest average power (99.61% of the original design). The slightly larger overheads on the NVDLA_partition_m/p reflect the cost of aggressive timing repair for severe violations. Overall, the QoR results under a common preference that prioritizes timing before power and area demonstrate that our agentic framework is stronger for iterative

Table 2: Customized optimization with allowable WNS degradation of 5% of the clock period. The power and area reductions are compared with Ours in Table 1. Additional power-breakdown comparisons are in Section B.

Design	Allowable WS (ps)	Achieved WS (ps)	Power Reduction	Area Reduction
ariane136	-35	-34.90	0.05%	0.35%
mempool tile_wrap	-130	-130.00	1.90%	0.21%
aes_256	-27.5	-27.49	1.34%	1.85%
hidden1	-30	-14.23	0.00%	0.11%
hidden2	-35	-34.29	0.00%	0.06%

ECO optimization than RL-based black-box search, even though RL observes more EDA feedback traces during training.

We further present the hypervolume progression during the optimization process of the Pareto frontier over WNS, power, and area for selected design cases in Figure 4. We set the area and power reference points to $1.1\times$ their initial values to avoid zero hypervolume since optimizing timing may slightly degrade power and area. Our agentic framework reaches a larger hypervolume within 10 rounds, whereas PPO [28] and A3C [26] require many more training episodes and still converge to a lower hypervolume. This result indicates that our framework not only satisfies the default timing-first preference, but also explores a stronger Pareto frontier for comprehensive QoR trade-offs.

5.3 Customized Optimization with User Preference

In practice, engineers often optimize under user-specified preferences, such as trading limited timing margin for lower power or area [16, 29]. Such preference-driven objectives are difficult to realize with black-box optimizers (e.g., RL) without careful reward engineering [9]. In contrast, LLM-based agents can directly interpret natural-language preferences and adjust scheduling accordingly.

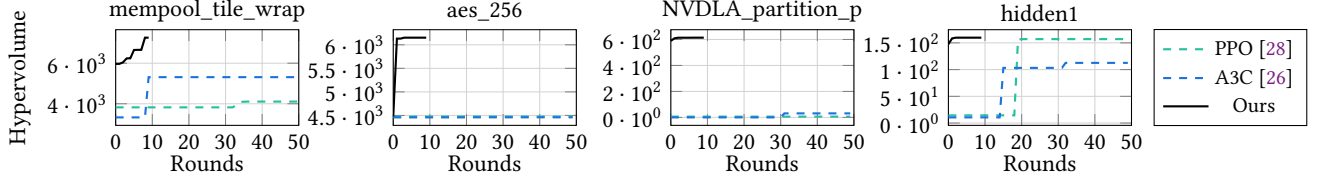


Figure 4: Hypervolume progression for selected design cases. PPO and A3C are plotted over 50 training episodes, while our agent is plotted over 10 rounds.

Table 3: Ablation results averaged over all designs in Table 1. The best results are bolded.

Method	Total Slack (ps)	Worst Slack (ps)	Power (W)	Area (μm^2)
GPT-5-mini	-8.441e+02	-1.326e+01	2.559e-01	1.981e+04
GPT-4o-mini	-1.534e+03	-2.536e+01	2.581e-01	2.039e+04
DeepSeek-V3.2	-1.670e+03	-2.195e+01	2.557e-01	1.977e+04
w/o RAG, w/ Reflect	-1.559e+03	-2.668e+01	2.581e-01	1.992e+04
w/ RAG, w/o Reflect	-1.829e+03	-2.371e+01	2.558e-01	1.986e+04
w/o RAG, w/o Reflect	-1.973e+04	-2.716e+01	2.566e-01	2.005e+04

We select five designs from Table 1 that achieve timing-clean solutions under the default preference, and then relax timing by allowing WNS degradation up to 5% of the clock period. Table 2 reports the allowable/achieved worst slack and the power/area reductions relative to **Ours** in Table 1.

Across all five designs, the achieved worst slacks satisfy the allowable limits. Power reductions are near-zero on hidden1 and hidden2 because clock power dominates total power, limiting overall reduction. When excluding clock power, the customized objective yields more substantial reductions, as analyzed in Section B. Overall, these results show that our framework can follow language-instructed preferences and satisfies customized QoR objectives during ECO.

5.4 Ablation Study

We perform ablation studies on the RAG and reflection modules of our framework and test the framework with different backbone LLMs. Results averaged over all designs are reported in Table 3. Replacing the default GPT-5-mini with GPT-4o-mini (2024-07-18) yields a modest degradation in both slack metrics. We also use open-source DeepSeek-V3.2 (2025-12-01) [30], which improves power and area slightly, but results in slightly worse timing slack.

Removing the RAG and reflection mechanisms individually results in QoR degradation, with the removal of both leading to the worst results. Thus, both natural-language expertise grounding and QoR-aware reflection are essential for a successful ECO.

5.5 Runtime and Cost Analysis

We report average token usage, API cost, and runtime in Table 4. The RL baselines run for over 50 episodes (rounds) but still achieve lower QoR than our method with 10 rounds. We therefore directly compare the end-to-end runtime of RL over 50 episodes with our framework over 10 rounds. Under this setting, our full method finishes in 3.54h, while training RL takes 15.38h, giving a 4.3 $\times$ runtime reduction. This speed advantage comes in addition to the QoR gains reported above, showing that language-guided iterative optimization is more efficient than black-box RL for ECO flows.

Removing RAG and Reflection components reduces token usage and runtime, but also weakens QoR. Without RAG, the average token count drops from 1.45M to 0.58M and runtime decreases from

Table 4: Runtime and cost analysis across methods averaged over all designs.

Method	Token	Cost (\$)	Runtime (h)
A3C [26]/PPO [28]	–	–	15.38/17.45
Ours	1,451,290	1.25	3.54
Ours (GPT-4o-mini)	911,470	0.16	0.93
Ours (DeepSeek-V3.2)	1,003,061	1.16	2.61
Ours w/o RAG, w/ Reflect	582,528	0.46	1.55
Ours w/ RAG, w/o Reflect	1,393,026	1.18	2.83
Ours w/o RAG, w/o Reflect	544,610	0.44	1.48

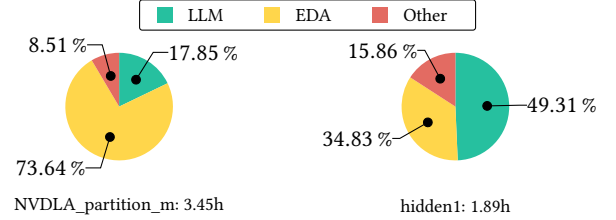


Figure 5: Runtime breakdown for NVDLA_partition_m and hidden1. The *Other* sector denotes other runtime costs like EDA tool report parsing.

3.54h to 1.55h; removing both RAG and reflection further reduces the cost to 0.54M tokens and 1.48h. Even with these components, the full framework remains much faster than RL and incurs modest token cost, making it practical for ECO flows.

Figure 5 shows that the remaining runtime is largely due to EDA tool execution, especially on large designs. For the larger NVDLA case, EDA calls account for 73.6% of total runtime, compared with 17.9% for LLM calls and 8.5% for other overhead. Thus, for industrial large designs, LLM interaction is mostly amortized by expensive EDA runs, and the agent acts as a preference-aware scheduling layer rather than the dominant runtime bottleneck.

6 Conclusion

This paper presents **Retrieve-Schedule-Reflect**, an agentic LLM framework that improves chip QoR by scheduling iterative ECO actions through EDA tools. Our framework uses search-tree retrieval to ground scheduling decisions in natural-language expert knowledge, executes ECO actions through EDA tools, and applies Pareto-driven reflection to improve timing-power-area trade-offs across rounds. Experiments on eight designs show that our method achieves stronger timing optimization than reinforcement-learning baselines while reducing power and area, with more than 4 $\times$ runtime speedup. The framework also supports user-defined optimization preferences expressed in natural language, enabling customized trade-offs without task-specific reward design.

References

- [1] Y. Pu, Y. Ji, S. Yu, Z. Zhang, Z. Guo, Z. He, Y. Lin, D. Z. Pan, and B. Yu, "Gpu acceleration for versatile buffer insertion," in *Proc. ICCAD*, 2025.
- [2] Y.-C. Lu, S. Nath, V. Khandelwal, and S. K. Lim, "Rl-sizer: Vlsi gate sizing for timing optimization using deep reinforcement learning," in *Proc. DAC*. IEEE, 2021, pp. 733–738.
- [3] Y.-C. Lu, S. Nath, S. Pentapati, and S. K. Lim, "Eco-gnn: Signoff power prediction using graph neural networks with subgraph approximation," *ACM Transactions on Design Automation of Electronic Systems*, vol. 28, no. 4, pp. 1–22, 2023.
- [4] W. Ning, "Strongly np-hard discrete gate-sizing problems," *IEEE transactions on computer-aided design of integrated circuits and systems*, vol. 13, no. 8, pp. 1045–1051, 1994.
- [5] W. L. Neto, Y. Li, P.-E. Gaillardon, and C. Yu, "Flowtune: End-to-end automatic logic optimization exploration via domain-specific multiarmed bandit," *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, vol. 42, no. 6, pp. 1912–1925, 2022.
- [6] A. Agnesina, K. Chang, and S. K. Lim, "Vlsi placement parameter optimization using deep reinforcement learning," in *Proc. ICCAD*, 2020, pp. 1–9.
- [7] Y.-C. Lu, W.-T. Chan, D. Guo, S. Kundu, V. Khandelwal, and S. K. Lim, "Rl-ccd: concurrent clock and data optimization using attention-based self-supervised reinforcement learning," in *Proc. DAC*. IEEE, 2023, pp. 1–6.
- [8] D. Won and T. Kim, "Eda tool parameter optimization through fast and reliable machine-learning based rank prediction model," in *Proc. MLCAD*. IEEE, 2025, pp. 1–7.
- [9] R. Rafailov, A. Sharma, E. Mitchell, C. D. Manning, S. Ermon, and C. Finn, "Direct preference optimization: Your language model is secretly a reward model," *Advances in neural information processing systems*, vol. 36, pp. 53 728–53 741, 2023.
- [10] L. T. Clark, V. Vashishtha, L. Shifren, A. Gujja, S. Sinha, B. Cline, C. Ramamurthy, and G. Yeric, "Asap7: A 7-nm finfet predictive process design kit," *Microelectronics Journal*, vol. 53, pp. 105–115, 2016.
- [11] J. Fishburn and A. Dunlop, "A polynomial programming approach to transistor sizing," in *Proc. ICCAD*, 1985, pp. 326–328.
- [12] W. Shi and Z. Li, "A fast algorithm for optimal buffer insertion," *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, vol. 24, no. 6, pp. 879–891, 2005.
- [13] L. P. Van Ginneken, "Buffer placement in distributed rc-tree networks for minimal elmore delay," in *Proc. ISCAS*. IEEE, 1990, pp. 865–868.
- [14] T. Ajayi and D. Blaauw, "Openroad: Toward a self-driving, open-source digital layout implementation tool chain," in *Proceedings of Government Microcircuit Applications and Critical Technology Conference*, 2019.
- [15] Synopsys, "Primeclosure: Ai-driven signoff eco solution," <https://www.synopsys.com/implementation-and-signoff/signoff/primeclosure.html>, 2022, accessed: 2026-01-26.
- [16] O. Coudert, "Timing and design closure in physical design flows," in *Proc. ISQED*. IEEE, 2002, pp. 511–516.
- [17] M. Liu, T.-D. Ene, R. Kirby, C. Cheng, N. Pinckney, R. Liang *et al.*, "Chipnemo: Domain-adapted llms for chip design," *arXiv preprint arXiv:2311.00176*, 2023.
- [18] Z. He, H. Wu, X. Zhang, X. Yao, S. Zheng, H. Zheng, and B. Yu, "Chateda: A large language model powered autonomous agent for eda," *arXiv preprint arXiv:2308.10204*, 2023.
- [19] A. Ghose, A. B. Kahng, S. Kundu, and Z. Wang, "Orfs-agent: Tool-using agents for chip design optimization," in *Proc. MLCAD*, 2025.
- [20] S. Yao, J. Zhao, D. Yu, N. Du, I. Shafran, K. Narasimhan, and Y. Cao, "React: Synergizing reasoning and acting in language models," *arXiv preprint arXiv:2210.03629*, 2023.
- [21] P. Lewis, E. Perez, A. Piktus, F. Petroni, V. Karpukhin, N. Goyal, H. Küttler, M. Lewis, W.-t. Yih, T. Rocktäschel *et al.*, "Retrieval-augmented generation for knowledge-intensive nlp tasks," *Advances in neural information processing systems*, vol. 33, pp. 9459–9474, 2020.
- [22] N. Shinn, F. Cassano, A. Gopinath, K. Narasimhan, and S. Yao, "Reflexion: Language agents with verbal reinforcement learning," *Advances in Neural Information Processing Systems*, vol. 36, pp. 8634–8652, 2023.
- [23] Z. Pei, F. Liu, Z. He, G. Chen, H. Zheng, K. Zhu, and B. Yu, "Alphasyn: Logic synthesis optimization with efficient monte carlo tree search," in *Proc. ICCAD*. IEEE, 2023, pp. 1–9.
- [24] J. Wei, X. Wang, D. Schuurmans, M. Bosma, F. Xia, E. Chi, Q. V. Le, D. Zhou *et al.*, "Chain-of-thought prompting elicits reasoning in large language models," *Advances in neural information processing systems*, vol. 35, pp. 24 824–24 837, 2022.
- [25] H. Geng, T. Chen, Y. Ma, B. Zhu, and B. Yu, "Ptp: Physical design tool parameter tuning via multi-objective bayesian optimization," *IEEE transactions on computer-aided design of integrated circuits and systems*, vol. 42, no. 1, pp. 178–189, 2022.
- [26] C. Bai, J. Zhai, Y. Ma, B. Yu, and M. D. Wong, "Towards automated risc-v microarchitecture design with reinforcement learning," in *Proc. AAAI*, vol. 38, no. 1, 2024, pp. 12–20.
- [27] B.-Y. Wu, R. Liang, G. Pradipta, A. Agnesina, H. Ren, and V. A. Chhabria, "2024 iccad cad contest problem c: Scalable logic gate sizing using ml techniques and gpu acceleration," in *Proc. ICCAD*, 2024, pp. 1–5.
- [28] H.-H. Hsiao, Y.-C. Lu, P. Vanna-Iampikul, and S. K. Lim, "A hybrid reinforcement learning framework for efficient physical design parameter tuning," *ACM Transactions on Design Automation of Electronic Systems*, 2026.
- [29] T. Sharma, S. Kolluru, and K. S. Stevens, "Learning based timing closure on relative timed design," in *Proc. VLSI-SoC*. Springer, 2020, pp. 133–148.
- [30] A. Liu, A. Mei, B. Lin, B. Xue, B. Wang, B. Xu, B. Wu, B. Zhang, C. Lin, C. Dong *et al.*, "DeepSeek-V3.2: Pushing the Frontier of Open Large Language Models," *arXiv preprint arXiv:2512.02556*, 2025.

A Scheduling Search Tree

Each search-tree node v associates an objective $o \in \{\text{timing, power, area}\}$ with a natural-language design summary s , an EDA-tool strategy $\mathcal{G}$, and a relevant manual excerpt (licensed text omitted). We construct the tree from literature, expert ECO strategies, and tool manuals, then apply expert filtering to remove invalid transitions; for example, timing optimization should not change a “small” violation into a “significant” one. Tree paths form scheduling templates that encode state transitions, objective preferences, and tool-use strategies. At each schedule–execute iteration, the scheduler retrieves paths matching the current state, QoR changes, and user preference, as illustrated below.

SCHEDULING PATH TEMPLATE EXAMPLE

Timing (Aggressive) → Timing (Incremental) → Power → Timing (Incremental) ****Design summary****: The design exhibits severe timing violations. After aggressive timing optimization, several violations remain. We then perform incremental timing optimization with a different algorithm. After they are fixed, we perform power recovery. The power-recovery step may introduce timing degradation that requires incremental timing optimization.

****Target selection****: Use an aggressive timing pass to gain slack fast, then an incremental timing pass to stabilize, then optimize power, and end with incremental timing cleanup. Switch away from timing if it becomes well-exploited with no gains.

****Option selection****: Aggressive timing is ****exploration-heavy**** (broad option coverage) to expose limits. Incremental timing is ****exploitation-heavy****, using only options shown to help and avoiding prior no-effect attempts. Power stage starts with exploration to find which power components can be reduced safely, then exploits the best component-focused options.

****Guidelines****: Don't chase truly non-fixable timing violations. If aggressive timing increases power/area, explicitly recover power afterward (and note whether area needs a later pass). After power changes, always run a timing cleanup to repair any regressions while minimizing new side-effects.

B Additional Experimental Results with Customized Optimization Tasks

Table 5: Original ASAP7 design statistics.

Metric	NVDLA_partition_m	NVDLA_partition_p	ariane136	mempool_tile_wrap	aes_256	hidden1	hidden2	hidden5
Clock period (ps)	4e+02	4e+02	7e+02	2.6e+03	5.5e+02	7e+02	7e+02	3e+02
Total slack (ps)	-1.909e+04	-2.073e+05	-2.189e+05	-2.133e+06	-5.774e+04	-4.237e+04	-1.325e+05	-1.512e+05
Worst slack (ps)	-3.657e+02	-2.784e+02	-1.667e+02	-4.777e+02	-5.820e+01	-1.314e+02	-1.823e+02	-8.691e+01
Area (μm^2)	2.728e+03	9.570e+03	3.266e+04	2.813e+04	2.728e+04	5.355e+03	3.307e+04	2.734e+04
Power (W)	5.250e-03	5.450e-02	5.928e-01	4.640e-02	3.037e-01	2.200e-02	5.863e-01	4.440e-01

Table 5 summarizes the original designs. For customized optimization, we allow up to 5% worst-slack degradation and report only combinational and register power: clock-network power dominates but is not the benchmark focus [27], while area has no cell-level breakdown. Tables 6 and 7 compare the default (“Ours”) and customized (“Custom”) objectives. Customized optimization reduces combinational power most strongly because sizing and buffering primarily affect combinational logic. The zero register power of ariane136 and hidden2 results from asynchronous flip-flops not classified as standard registers; the remaining designs still show register-power reductions. These results show that the agents interpret user objectives and schedule optimizations for customized trade-offs.

Table 6: Combinational power comparison; reduction is $(\text{Ours} - \text{Custom})/\text{Ours} \times 100\%$.

Design	Total (W)			Leak (W)			Dyn (W)		
	Ours	Custom	Reduction	Ours	Custom	Reduction	Ours	Custom	Reduction
aes_256	2.39e-1	2.32e-1	2.9%	1.34e-4	1.07e-4	20.1%	2.39e-1	2.32e-1	2.9%
ariane136	4.16e-2	3.29e-2	20.9%	4.31e-5	1.74e-5	59.6%	4.15e-2	3.29e-2	20.7%
hidden1	2.22e-3	2.14e-3	3.6%	1.95e-5	1.07e-5	45.1%	2.20e-3	2.13e-3	3.2%
hidden2	3.68e-2	3.63e-2	1.4%	1.32e-5	1.25e-5	5.3%	3.68e-2	3.63e-2	1.4%
mempool_tile_wrap	1.80e-2	1.74e-2	3.3%	1.30e-3	1.13e-3	13.1%	1.68e-2	1.62e-2	3.6%

Table 7: Register power comparison; reduction is $(\text{Ours} - \text{Custom})/\text{Ours} \times 100\%$; “–” is non-comparable.

Design	Total (W)			Leak (W)			Dyn (W)		
	Ours	Custom	Reduction	Ours	Custom	Reduction	Ours	Custom	Reduction
aes_256	4.23e-2	4.10e-2	3.1%	4.53e-6	4.53e-6	0.0%	4.23e-2	4.10e-2	3.1%
ariane136	0	0	–	0	0	–	0	0	–
hidden1	4.63e-4	4.61e-4	0.4%	2.33e-6	2.33e-6	0.0%	4.60e-4	4.59e-4	0.2%
hidden2	0	0	–	0	0	–	0	0	–
mempool_tile_wrap	5.25e-4	5.15e-4	1.9%	2.45e-6	2.39e-6	2.4%	5.23e-4	5.13e-4	1.9%