Dr. RTL: Autonomous Agentic RTL Optimization through Tool-Grounded Self-Improvement

Wenji Fang, Yao Lu, Shang Liu, Jing Wang, Ziyan Guo,
Junxian He, Fengbin Tu, Zhiyao Xie*
Hong Kong University of Science and Technology

ABSTRACT

Recent advances in large language models (LLMs) have sparked growing interest in automatic RTL optimization for better performance, power, and area (PPA). However, existing methods are still far from realistic RTL optimization: their evaluation is often unrealistic, based on manually degraded, small-scale RTL designs and weak open-source toolchains; their methods are also limited, relying on coarse design-level feedback and simple pre-defined rewriting rules. To address these limitations, we present Dr. RTL, an agentic framework for RTL timing optimization in a realistic evaluation environment, with continual self-improvement through reusable optimization skills. We establish a realistic evaluation setting with more challenging RTL designs and an industrial EDA workflow. Within this setting, Dr. RTL performs closed-loop optimization through a multi-agent framework for critical-path analysis, parallel RTL rewriting, and tool-based evaluation. We further introduce group-relative skill learning, which compares parallel RTL rewrites and distills the optimization experience into an interpretable skill library. Currently, this library contains 47 pattern–strategy entries for cross-design reuse to improve PPA and accelerate convergence, and it can continue evolving over time. Evaluated on 20 real-world RTL designs, Dr. RTL achieves average WNS/TNS improvements of 21%/17% with a 6% area reduction over the industry-leading commercial synthesis tool.

1 INTRODUCTION

Register-transfer level (RTL) design is a critical abstraction in modern digital ICs, manually developed by human engineers to bridge architectural specification and downstream logic synthesis. The quality of RTL largely determines the optimization space available to the subsequent synthesis and physical design tools. Modern RTL designs often incorporate deep pipelines, wide arithmetic datapaths, and tightly coupled control–data interactions [36], all of which make timing closure more challenging. Bottlenecks introduced at the RTL stage often propagate through the design flow, leading to repeated RTL revisions, conservative timing constraints, and suboptimal PPA trade-offs. Consequently, improving RTL quality before synthesis remains an important yet challenging task.

Automatic synthesis optimization vs. manual RTL optimization. Synthesis and RTL optimization operate at different abstraction levels and complement each other. Synthesis improves PPA by transforming a fixed RTL into an optimized gate-level implementation through Boolean rewriting and technology mapping [4, 6, 9, 15, 18, 40, 46], but it remains constrained by the original RTL structure and cannot change higher-level design choices such as datapath organization or pipelining. Consequently, further

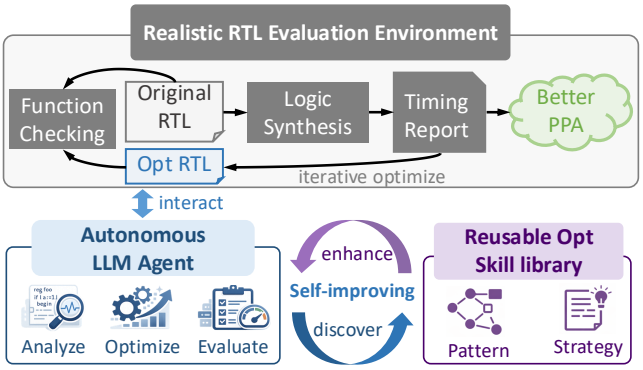


Figure 1: Dr. RTL iteratively optimizes RTL PPA via closed-loop interaction with industrial EDA tools, while distilling trajectories into reusable skills for self-improvement.

PPA improvement often requires iterative manual RTL rewriting guided by synthesis feedback: engineers identify critical paths with timing reports, revise the RTL, and re-run synthesis to evaluate each change, as illustrated in Figure 1. However, this process remains labor-intensive, time-consuming, and heavily dependent on implicit human expertise, making it difficult to automate systematically. More fundamentally, the RTL design space is vast, effectively infinite for practical purposes, while human-driven optimization typically evaluates only one candidate design per iteration, exploring only a negligible fraction of the possible optimization space.

LLM for RTL Design. To reduce this manual effort, recent advances in large language models (LLMs) have sparked growing interest in RTL design automation, as summarized in recent surveys [7, 13, 16, 25, 29, 42]. Most prior works [1, 2, 8, 10–12, 14, 17, 20, 21, 24, 26, 27, 30, 33–35, 41, 43–45] focus on generating RTL from natural language specifications, with functional correctness as the primary objective and PPA only weakly considered. They are also typically evaluated on small-scale benchmarks such as VerilogEval [19] and RTLLM [22], which fall short of the complexity and critical-path behavior of real-world RTL designs. More recently, several studies [5, 23, 28, 37–39] start to explore *RTL optimization*, aiming to rewrite existing RTL for better PPA while preserving functionality, as summarized in Table 1.

Limitations of existing LLM-based RTL optimization. Despite recent progress, existing approaches remain far from addressing realistic RTL optimization in both evaluation and methodology:

From an *evaluation* perspective, they suffer from three key issues: *(1) Weak starting RTL.* Prior works [5, 23, 28, 37–39] evaluate on manually degraded RTL with injected degradation or redundancies. This makes the task easier and less realistic, reducing it to PPA repair rather than true optimization of well-written RTL. *(2) Unrealistic toolchains.* Most studies [5, 28, 37–39] rely on open-source synthesis tools that are much weaker than commercial ones, so some reported gains may already be absorbed by commercial tools and may reflect tool limitations rather than true RTL optimization. *(3) Very limited scale.* Existing evaluations [5, 23, 28, 37–39] focus on tiny modules and fail to reflect the scale and complexity of industrial RTL designs.

*Corresponding Author (eezhiyao@ust.hk)

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

ACM ISBN 979-8-4007-2873-0/2026/11
https://doi.org/10.1145/3831252.3833975

Table 1: Comparison with existing LLM-based RTL optimization benchmarks and methods.

Work	RTL Dataset				Tool Use		Planning	Memory
	LoC⋆	NoM⋆	Complexity	Opt Input	EDA Tool	PPA Feedback		
RTLRewriter [39]	[8, 86, 1275]	[1, 1, 14]	Module	Manual-degraded RTL	Yosys + Simulation	Coarse-grained design PPA only	Single LLM	Pre-defined rewriting rules
SymRTLo [37]	Same as [39]				Yosys/DC+ Simulation			
RTL-OPT [23]	[8, 31, 135]	[1, 1, 1]			DC + CEC†			
POET [28]	Same as [23]				Yosys + Simulation			
CODMAS [5]	Not publicly accessible				Yosys + Simulation		Multi-LLM	
Dr. RTL (ours)	**[128, 812, 4615]**	**[1, 3, 7]**	**IP/Design**	**Original human-written RTL**	**DC + SEC†**	**Fine-grained critical path**	**Orchestrator + Sub-agents**	**Skill self-improving**

⋆ We report the [minimum, average, maximum] lines of code (LoC) and number of modules (NoM) for each dataset. Our dataset is around 10× larger than existing ones.
† Prior work relies on combinational equivalence checking (CEC), which cannot verify sequential changes. Our method employs sequential equivalence checking (SEC), supporting both combinational and sequential optimization scenarios.

From a *methodology* perspective, existing methods are limited by two factors. First, they rely mainly on LLM-based code inspection, without fine-grained EDA feedback to guide critical-path optimization. Second, they depend on pre-defined basic rewriting rules from textbooks [37, 39], which are often ineffective under modern synthesis tools and restrict the discovery of non-trivial optimization strategies. These gaps motivate the following research questions:

- **Q1. Evaluation:** How should RTL optimization be evaluated to reflect the realistic industrial scenario?
- **Q2. Method:** How can we perform effective RTL optimization and discover reusable optimization knowledge?
- **Q3. Optimization Gap:** What optimization opportunities cannot be handled by industrial synthesis and therefore require RTL optimization?

To address these questions, we propose **Dr. RTL**[1], a tool-grounded agentic RTL timing optimization framework with self-improving capability. As shown in Figure 1, built on a realistic evaluation setting (**A1**), Dr. RTL performs closed-loop optimization (**A2**) through iterative interaction with EDA tools while continually distilling reusable optimization knowledge into a skill library (**A3**).

We first answer Q1: **A1. Realistic industrial evaluation.** We establish a realistic evaluation environment for RTL timing optimization with 20 larger-scale, human-written designs. Compared with prior works, it improves over three aspects: *1) Realistic starting RTL,* starting from human-written RTL instead of manually degraded inputs, reflecting true optimization scenario; *2) Industrial-standard toolchain,* using commercial synthesis for strong optimization and sequential equivalence checking (SEC) to ensure both combinational and sequential equivalence, ensuring high-fidelity PPA evaluation; and *3) Larger design scale,* evaluating designs that are around 10× larger and more complex than prior small-scale designs, better capturing the hierarchy and critical-path complexity of real RTL.

Under this evaluation, we answer Q2 with **A2. Tool-grounded self-improving agentic RTL optimization.** We propose two coupled techniques for effective RTL optimization and continual self-improvement: *(1) Agentic optimization:* it formulates RTL optimization as a tool-grounded search problem over equivalent RTL transformations, using a multi-agent framework to iteratively perform timing analysis, RTL candidate rewriting, and tool-based evaluation. This enables adaptive exploration and exploitation over a large optimization space guided by fine-grained critical-path feedback. *(2) Group-relative skill learning:* Dr. RTL compares parallel rewritten RTL candidates from the same parent RTL and distills their optimization trajectories into reusable, human-interpretable skills, enabling continual test-time self-improvement without supervision or parameter tuning.

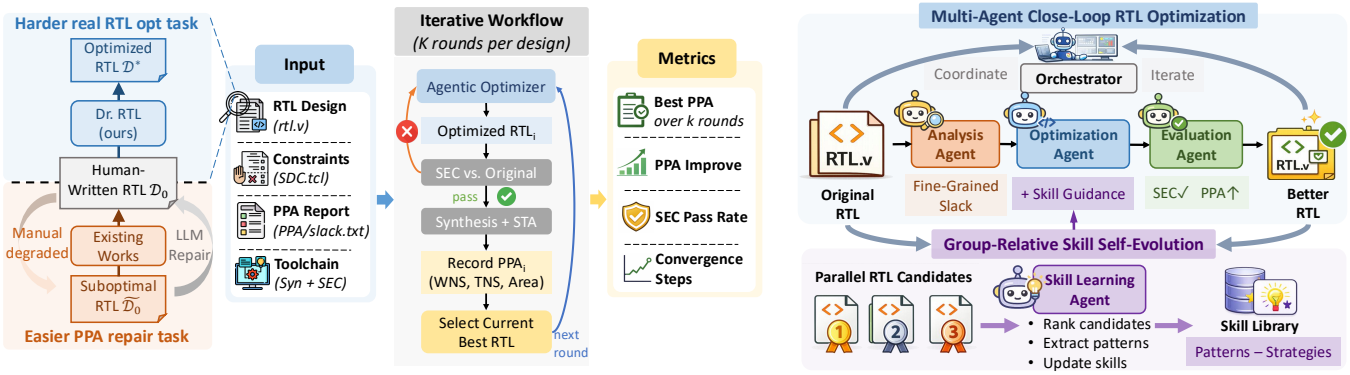
We further answer Q3 with **A3. Reusable RTL optimization knowledge.** The learned skills capture three forms of reusable knowledge, enabling cross-design reuse for better PPA, fewer invalid transformations, and faster convergence. Specifically, *(1) Timing bottleneck patterns:* realistic RTL timing bottlenecks are often recurrent, arising from structural patterns such as deep decode logic, wide comparisons, and mux-heavy selection, which reveal optimization opportunities beyond synthesis alone; *(2) Effective strategies:* for these patterns, effective strategies consistently emerge, including pre-computation, decomposition, and selective register insertion; and *(3) Invalid strategies:* many naive transformations are either absorbed by synthesis or violate equivalence, such as manual rebalancing of optimized logic or moving control updates across registers. In our current implementation, Dr. RTL extracts 47 pattern–strategy entries and releases them as a public skill library, which can be further expanded by applying the framework to more designs and incorporating expert knowledge from RTL designers.

Together, Dr. RTL takes a step toward practical agentic RTL optimization that improves at test time without requiring LLM fine-tuning. More broadly, it points to a new paradigm of *agentic design automation*, in which humans define the task, VLSI workflow, and constrained EDA feedback, while the agent autonomously explores effective solutions and accumulates reusable knowledge within this environment for self-improvement.

The contributions of this work are summarized as follows:

- **Problem.** We identify RTL timing optimization as a practical and underexplored problem, where strong human-written RTL must be improved beyond synthesis rather than merely repaired from degraded inputs.
- **Evaluation.** We establish a realistic evaluation for RTL optimization, featuring challenging human-written RTL, commercial synthesis, and sequential formal verification.
- **Method.** We develop Dr. RTL, a tool-grounded closed-loop agentic RTL optimization framework, together with group-relative skill learning for continual self-improvement.
- **Knowledge.** We extract explicit RTL optimization knowledge, consisting of 47 pattern–strategy pairs, for cross-design reuse to improve PPA and accelerate convergence.
- **Result.** Across 20 diverse RTL designs, Dr. RTL achieves average WNS/TNS improvements of 21%/17% while reducing area by 6% over commercial synthesis, demonstrating Pareto optimization beyond simple timing–area trade-offs.

[1]The RTL dataset, evaluation environment, agentic implementation, and learned optimization skills are open-sourced at https://github.com/hkust-zhiyao/Dr_RTL.



(a) Evaluating agentic RTL timing optimization (Sec. 3) (b) Overview of Dr. RTL agentic framework (Sec. 4)

Figure 2: Proposed industrial-standard RTL optimization evaluation and overview of Dr. RTL for agentic RTL optimization.

2 PROBLEM FORMULATION

We formulate RTL timing optimization as an *agentic optimization problem*, where the LLM agent iteratively interacts with a commercial EDA environment. The synthesis and verification flow evaluates candidate RTL designs, and the agent proposes transformations based on the resulting feedback. Formally, given an initial RTL design $\mathcal{D}_0$, the goal is to obtain an optimized design $\mathcal{D}^*$ that improves post-synthesis PPA while preserving functional equivalence. We assume fixed synthesis settings and preserve the original micro-architecture, including pipeline latency. Under this constraint, we allow both combinational transformations and latency-preserving sequential restructuring, such as retiming-like register redistribution or duplication. The optimization objective is defined as:

$$\min_{\mathcal{D}} f(\text{WNS}, \text{TNS}, \text{Area}) \quad \text{s.t.} \quad \mathcal{D} \equiv \mathcal{D}_0. \tag{1}$$

For PPA evaluation, we focus on timing optimization while controlling area as a trade-off. Power is excluded from the current objective because accurate power estimation is workload-dependent and cannot be reliably assessed through static evaluation alone.

3 AGENTIC RTL OPTIMIZATION EVALUATION ENVIRONMENT

Figure 2(a) illustrates our industrial-standard evaluation environment for agentic RTL timing optimization. Compared with prior benchmarks [5, 23, 39], our setup improves over three dimensions: input designs, toolchain, and evaluation metrics.

(1) Input designs. We evaluate on original human-written RTL designs $\mathcal{D}_0$ collected from diverse open-source projects, rather than manually degraded RTL $\tilde{\mathcal{D}}_0$ constructed by injecting artificial degradations or redundancies into a reference design $\mathcal{D}_0$. As summarized in Table 1, prior benchmarks are typically limited to small single-module designs, often with only tens of lines of code. In contrast, our dataset contains substantially larger and more complex designs, with 812 lines of code on average, deeper module hierarchies, and diverse design types and coding styles. This setting better reflects realistic RTL optimization to improve already strong human-written RTL.

(2) Commercial synthesis with complete formal verification. We adopt an industrial-standard EDA workflow that combines commercial logic synthesis with sequential formal equivalence checking (SEC). Commercial synthesis provides stronger and more realistic optimization than open-source tools such as Yosys [5, 28, 37, 39], reducing the chance that simple RTL rewrites appear effective only because the synthesis is weak. SEC is used to ensure equivalence after both combinational and sequential transformations. This is stronger than simulation-based checking [5, 28, 37, 39], which is incomplete, and combinational equivalence checking [23], which cannot support sequential rewrites. Together, this workflow enables reliable evaluation of RTL optimization under realistic industrial constraints.

(3) Agentic optimization metrics. To evaluate iterative agentic optimization, we report not only final PPA but also iteration metrics. Specifically, we measure: 1) *best PPA* over all iterations, reflecting the peak optimization result; 2) *PPA improvement* relative to the initial design $\mathcal{D}_0$; 3) *SEC pass rate*, reflecting the validity rate of proposed transformations; and 4) *convergence steps*, measuring how quickly the optimization stabilizes. Together, these metrics characterize both optimization quality and agentic behavior.

4 DR. RTL METHODOLOGY

Figure 2(b) shows the overall architecture of Dr. RTL. The framework combines two tightly coupled aspects: (1) a closed-loop optimization framework that iteratively performs timing analysis, RTL transformation, and tool-based evaluation under industrial EDA feedback via three specialized agents, as illustrated in Section 4.1, and (2) a learning mechanism based on group-relative skill learning, which compares parallel transformations under the same optimization context and distills effective experience into reusable pattern–strategy skills, as detailed in Section 4.2. Together, they enable continual improvement in both optimization quality and efficiency. We describe them below.

4.1 Agent Design and Responsibilities

Figure 3 illustrates our orchestrator–agent framework. The orchestrator coordinates three specialized agents: (1) a timing analysis agent that identifies critical-path bottlenecks from post-synthesis timing reports; (2) an RTL optimization agent that rewrites N candidate designs in parallel through equivalent transformations guided by the timing analysis; and (3) an evaluation agent that runs synthesis and equivalence checking to return PPA and correctness feedback for the next round of analysis.

This decomposition provides three benefits: (1) *task specialization*, where each agent focuses on a well-defined role aligned with the industrial workflow, improving reliability and interpretability; (2) *parallel exploration*, where parallel candidate generation and selection turn optimization into a tool-guided search over a non-convex design space; and (3) *modular learning*, where the decomposition enables optimization behaviors to be attributed, analyzed, and reused. Overall, this design aligns agent roles with tool

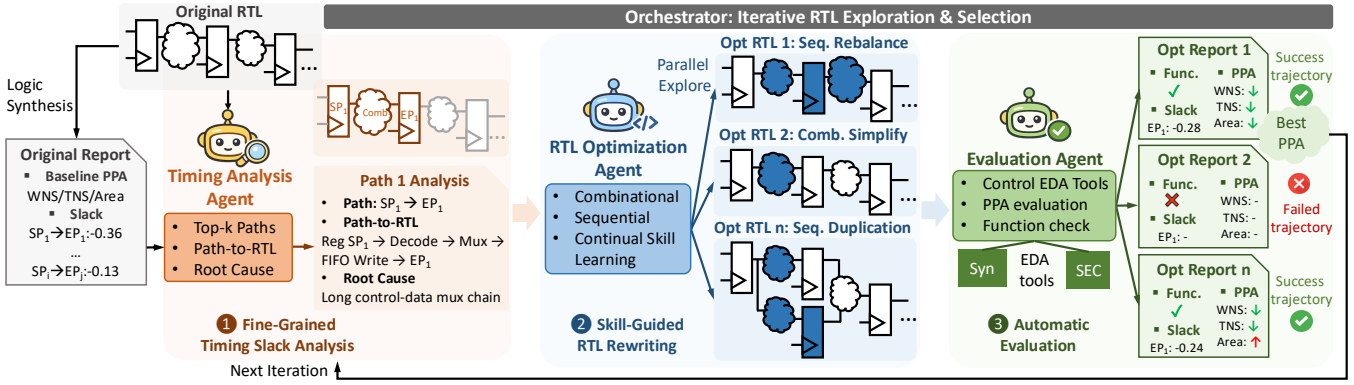


Figure 3: Multi-agent framework of Dr. RTL. Dr. RTL coordinates timing analysis, RTL optimization, and evaluation agents in a closed loop, using industrial EDA feedback to localize critical paths, explore RTL candidates in parallel, and promote the best SEC-passing design for iterative PPA improvement.

boundaries, leverages feedback, and balances exploration with refinement. We describe each agent below. The empirical benefit of the decomposition is shown in Section 5.5.

4.1.1 Orchestrator. The orchestrator has three responsibilities: executing the optimization loop, selecting among parallel candidates across iterations, and maintaining trajectory logs for subsequent skill learning. Starting from an initial design $\mathcal{D}_0$, the system evolves over $t = 0, 1, \ldots, K$ following

$$\mathcal{D}_t \rightarrow \{\mathcal{D}_t^{(i)}\}_{i=1}^{N} \rightarrow \mathcal{D}_{t+1}, \tag{2}$$

where agents generate and evaluate N candidate designs in parallel based on EDA tool feedback.

To coordinate the framework, the orchestrator maintains a shared JSON state that serves as the communication interface across agents. This state records timing analysis results, candidate RTL edits, evaluation outcomes, and iteration history, allowing each agent to consume structured outputs from previous stages and write back its own results. The same structured logs are also used to maintain optimization trajectories for subsequent skill learning. We provide a detailed illustration in Section 4.2.

To evaluate the PPA results, at each iteration, a scalar score is computed for each SEC-pass candidate:

$$Score_i = \alpha\,\mathrm{WNS}_i^{\mathrm{norm}} + \beta\,\mathrm{TNS}_i^{\mathrm{norm}} + \gamma\,\mathrm{Area}_i^{\mathrm{norm}} + \mathrm{penalty}_i, \tag{3}$$

where $\mathrm{PPA}_i^{\mathrm{norm}} = \frac{\mathrm{PPA}_i - \mathrm{PPA}_{\mathrm{baseline}}}{\mathrm{PPA}_{\mathrm{baseline}}}$, and $\alpha, \beta, \gamma \in [0, 1]$ are weighting factors[2]. Lower scores indicate better timing–area trade-offs. For WNS and TNS, improvement means moving closer to zero. The next starting design is chosen as the best SEC-passing candidate:

$$\mathcal{D}_{t+1} = \arg\min_{\mathcal{D}_t^{(i)}} Score_i, \quad \text{s.t.} \quad \mathrm{SEC}_i = 1, \tag{4}$$

where $\mathrm{SEC}_i \in \{0, 1\}$ indicates functional correctness.

4.1.2 Timing Analysis Agent. The timing analysis agent identifies timing bottlenecks through fine-grained slack analysis, but does not directly propose fixes. Rather than relying on coarse design-level feedback such as a single WNS value, it uses detailed post-synthesis timing reports to localize critical-path issues back to RTL structure.

Given the current design and timing report, the agent selects the top-k critical paths by slack. For each path, it performs path-to-RTL mapping by locating the startpoint and endpoint registers and tracing the intermediate combinational logic back to the corresponding RTL regions. It then diagnoses likely root causes of delay, such as excessive combinational depth, high fanout, control–data coupling, or reconvergent logic, to guide subsequent RTL optimization agent.

4.1.3 RTL Optimization Agent. The RTL optimization agent generates N candidate designs $\{\mathcal{D}_t^{(i)}\}_{i=1}^{N}$ in parallel based on timing analysis and the accumulated skill library $\mathcal{S}$ (detailed in Section 4.2). Guided by critical-path information and root-cause analysis, it applies equivalent transformations to handle true bottlenecks. It first queries $\mathcal{S}$ for pattern–strategy entries matching the diagnosed bottleneck. If a match is found, the corresponding skill-guided transformation is applied to $\mathcal{D}_t$. Otherwise, the LLM proposes a new transformation conditioned on the timing feedback.

This design balances *exploration* and *exploitation*. Exploration comes from generating multiple candidates in parallel, either by applying different transformations to the same bottleneck or by targeting different bottlenecks in the current iteration. Exploitation comes from promoting the best SEC-pass candidate $\mathcal{D}_{t+1}$ to the next iteration and from reusing effective skills in $\mathcal{S}$ across designs.

Compared with prior methods driven by coarse design-level metrics and fixed basic rewriting rules, our optimizer uses critical-path-level feedback and learned skills to guide adaptive RTL transformations, improving both search efficiency and optimization quality.

4.1.4 Evaluation Agent. The evaluation agent executes the synthesis and verification toolchain. Given a candidate design, it runs the commercial synthesis flow and sequential equivalence checking (SEC), and returns the resulting metrics (WNS_i, TNS_i, Area_i, SEC_i).

The evaluation agent performs no reasoning or report interpretation, ensuring a strict separation between execution and decision-making. This design guarantees that all optimization decisions are based solely on verified EDA feedback, avoiding hallucinations.

4.2 Group-Relative Skill Learning from Hierarchical Optimization Trajectories

A central goal of Dr. RTL is not only to optimize the current design, but also to accumulate reusable optimization knowledge over time. Without such skill learning, each iteration would rely mainly on local trial-and-error, making it difficult to systematically reuse successful transformations across iterations and designs. A reusable skill library is therefore important for improving PPA more efficiently, reducing invalid or unhelpful transformations, and accelerating convergence through cross-design reuse, and providing interpretable knowledge of what works and why.

[2]In our experiments, we set $\alpha = 0.5$, $\beta = 0.35$, and $\gamma = 0.15$. We set $\mathrm{penalty}_i = 0.5$ if $\mathrm{Area}_i^{\mathrm{norm}} > 0.1$, and 0 otherwise, to discourage excessive area overhead. Different settings can be used to target different PPA trade-offs.

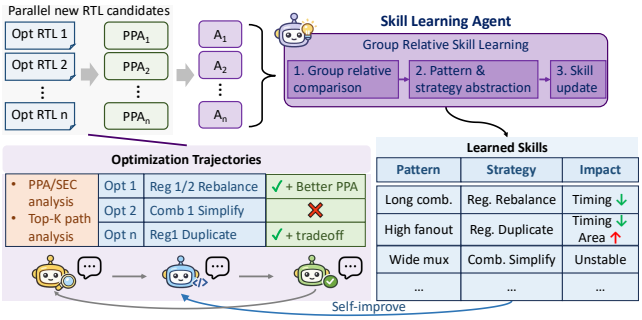


Figure 4: Group-relative skill learning. Parallel RTL candidates are compared to extract patterns and strategies, which are stored as reusable skills for continuous improvement.

A key challenge, however, is determining which transformations are truly worth storing as skills. Because EDA outcomes are heuristic and design-dependent, absolute PPA values are often noisy and not directly comparable across candidates or iterations. This makes it difficult to attribute improvement to any single transformation, especially when multiple edits are composed over time.

To address this, we propose *group-relative skill learning*, as illustrated in Figure 4. The core idea is to evaluate a strategy not by its absolute PPA outcome, but by how well it performs relative to other candidates generated from the same parent RTL under the same timing feedback and tool settings. We refer to the set of parallel candidates $\{\mathcal{D}_t^{(i)}\}_{i=1}^{N}$ in iteration t as a *group*. This within-group comparison provides a more stable signal for identifying effective strategies without external supervision.

Hierarchical optimization trajectory. Each candidate design is associated with an optimization trajectory that records its full optimization context, including the timing analysis, applied RTL transformations, and resulting evaluation outcomes. We organize these trajectories as a three-layer hierarchy. The top layer contains the iteration round, recording the temporal evolution of optimization. Within each round, the second layer stores all parallel candidate designs together with their PPA and SEC results, preserving the within-group context needed for relative comparison. At the lowest layer, it records critical-path-level information, including the analyzed bottlenecks, their structural patterns and root causes, the transformations applied, and the resulting outcomes. This hierarchical trajectory connects analysis, optimization actions, and evaluation results across multiple granularities, providing the structured context needed for comparison and skill extraction.

Group-relative comparison. Built on this trajectory representation, Dr. RTL compares parallel candidates within each group by computing a relative advantage signal:

$$A_i = \frac{score_i - \mu_t}{\sigma_t}, \tag{5}$$

where μ_t and σ_t denote the mean and standard deviation of the candidate scores (Eq. (3)) in iteration t. Intuitively, A_i measures whether a candidate performs better or worse than its peers under matched conditions, highlighting strategies that consistently outperform alternatives in the same optimization context.

Pattern–strategy abstraction. Guided by the relative advantage signal, we employ an auxiliary skill learning agent to analyze the trajectories and extract reusable pattern–strategy pairs as optimization skills. A pattern captures a recurring structural bottleneck on critical paths, such as deep FSM/decode logic, high-fanout control signals, wide comparisons, or mux-heavy selection logic. A strategy describes the corresponding transformation principle, such as condition pre-computation, signal replication, or selective register insertion. We will provide detailed case studies in Section 6.1.

Skill update. Based on these abstractions, Dr. RTL updates a confidence-aware skill library by merging newly extracted pattern–strategy pairs with existing entries and accumulating simple empirical statistics, including occurrence count, SEC-pass count, and mean relative advantage across iterations and designs. Skills that repeatedly appear in successful trajectories are assigned higher confidence, while unreliable or ineffective ones are deprioritized as invalid strategies.

Overall, group-relative skill learning converts noisy absolute PPA feedback into more reliable relative signals, enabling continuous self-improvement without supervision or reward design.

5 EXPERIMENTAL RESULTS

5.1 Experimental Setup

Diverse RTL design dataset. We evaluate Dr. RTL on 20 human-written Verilog designs selected using three criteria: complex functionality, full synthesizability, and size ranging from hundreds to thousands of lines of code. The dataset spans diverse IPs and systems, including buffers, processors, signal-processing modules, cryptographic designs, and SoCs, with a wide range of scales and structural complexity. Detailed statistics are summarized in Table 2, with an average of 812 LOC (min 128, max 4615) and 3 modules per design. We also report synthesized gate and register counts to reflect design complexity. For comparison, we further evaluate on all large designs from the representative benchmark of [39], detailed in Section 5.3.

Evaluation workflow. We evaluate Dr. RTL and all baselines under a unified protocol. All designs are synthesized with Synopsys Design Compiler using the Nangate 45 nm library [31] under fixed synthesis constraints for fair comparison. In particular, we use a tight clock period of 0.1 ns to force aggressive synthesis optimization across all paths. Functional equivalence between design versions is verified by sequential equivalence checking using Cadence Jasper SEC [32].

For each design, every method runs 10 iterations with 5 parallel candidates per iteration, for 50 optimization attempts in total. At each iteration, the best SEC-passing candidate is selected by PPA and promoted to the next round. We report the best PPA improvement, SEC pass rate, and optimization trajectories. To explicitly evaluate skill generalization, we use 4-fold cross-validation in the skill application setting: in each fold, optimization skills are extracted from 15 designs and then applied to 5 unseen designs. The skill library is reset for each fold, and all prompts and examples are kept strictly fold-isolated, preventing leakage from benchmark overlap. This ensures that the reported results reflect cross-design skill reuse rather than design-specific memorization.

Implementation of Dr. RTL. Dr. RTL is currently implemented on the Claude Code command-line interface (CLI), while the framework is agent-CLI agnostic and can be readily adapted to other agentic coding environments through the same markdown-based workflow specification. Unless otherwise noted, we use Claude Opus for the main experiments, and evaluate other Claude models such as Sonnet and Haiku in the scaling study to cover different

Table 2: PPA improvements of Dr. RTL over commercial logic synthesis on real-world RTL designs.

Design	Statistics				Commercial Synthesis			w/ Dr. RTL Optimization			
	LoC	NoM	#. Gate	#. Reg	WNS (*ns*)	TNS (*ns*)	Area (um^2)	WNS (*ns*)	TNS (*ns*)	Area (um^2)	SEC Pass
vending	128	1	20272	4	-0.27	-1.02	20488	-0.09 (-66.7%)	-0.5 (-51.0%)	20533 (0.2%)	77%
ticket	134	1	56	6	-0.23	-1.24	78	-0.09 (-60.9%)	-0.47 (-62.1%)	45 (-41.8%)	65%
lstm	135	4	8379	0	-6.51	-166.76	14828	-4.86 (-25.3%)	-116.16 (-30.3%)	4753 (-67.9%)	83%
dsp	165	5	3345	196	-2.61	-154.64	4755	-2.59 (-0.8%)	-147.42 (-4.7%)	4751 (-0.1%)	83%
communicate	225	3	1023	232	-0.4	-73.08	2092	-0.26 (-35.0%)	-58.84 (-19.5%)	2446 (17.0%)	95%
spi1	332	2	647	131	-0.32	-33.42	1208	-0.29 (-9.4%)	-33.53 (0.3%)	1276 (5.7%)	63%
cpu_fsm	354	1	13232	4163	-0.82	-429.73	32268	-0.61 (-25.6%)	-427.28 (-0.6%)	32157 (-0.3%)	96%
aes	374	2	24294	1419	-0.7	-913.49	33755	-0.67 (-4.3%)	-876.71 (-4.0%)	33975 (0.7%)	81%
fifo	390	7	13740	4208	-0.54	-2002.1	36061	-0.43 (-20.4%)	-1681.97 (-16.0%)	36310 (0.7%)	96%
spi2	441	3	856	292	-0.26	-28.19	1748	-0.25 (-3.8%)	-27.75 (-1.6%)	1826 (4.5%)	100%
uart	447	4	851	135	-0.38	-23.17	1272	-0.34 (-10.5%)	-23.08 (-0.4%)	1107 (-13.0%)	87%
controller	528	1	213	8	-0.38	-2.85	235	-0.33 (-13.2%)	-2.57 (-9.8%)	277 (18.0%)	94%
router	571	5	3036	609	-0.53	-289.91	5479	-0.46 (-13.2%)	-246.66 (-14.9%)	5575 (1.8%)	92%
cpu_pipe	850	5	2845	364	-0.38	-23.24	2622	-0.11 (-71.1%)	-2.69 (-88.4%)	2313 (-11.8%)	90%
pcie	923	7	1773	97	-0.79	-23.83	2156	-0.44 (-44.3%)	-19.75 (-17.1%)	1426 (-33.9%)	100%
datapth	1065	7	6985	881	-0.88	-513.42	12137	-0.87 (-1.1%)	-504 (-1.8%)	12137 (0.0%)	92%
i2c	1036	3	723	128	-0.36	-26.67	1290	-0.35 (-2.8%)	-25.96 (-2.7%)	1275 (-1.1%)	98%
tv80	4615	5	4431	359	-1.31	-381.2	6044	-1.22 (-6.9%)	-362.09 (-5.0%)	6200 (2.6%)	92%
arm_cpu1	2070	1	11772	1222	-5.24	-3148.68	22172	-5.19 (-1.0%)	-3117.41 (-1.0%)	22257 (0.4%)	56%
arm_cpu2	1450	1	6132	736	-1.01	-662.7	10688	-0.92 (-8.9%)	-619.13 (-6.6%)	10995 (2.9%)	87%
Avg.	**812**	**3**	**6230**	**760**	/			**-21.3%**	**-16.9%**	**-5.8%**	**86%**

Table 3: Comparison with existing single-shot LLMs and LLM-based RTL optimization methods.

Method		WNS	TNS	Area
Single-Shot LLM	Claude Opus	-2.4%	-2.8%	-0.7%
	GPT 5.3	-1.2%	0.3%	0.6%
SOTA Baselines⋆	RTLRewriter [39]	-4.9%	-6.3%	-3.1%
	SymRTLo [37]	-7.1%	-5.7%	-1.4%
Ours	**Dr. RTL**	**-21.3%**	**-16.9%**	**-5.8%**

⋆ We use the same backbone LLM, Claude Opus, for both the state-of-the-art LLM-based RTL optimization baselines and our Dr. RTL to ensure a fair comparison.

capability–cost trade-offs [3]. We also include OpenAI GPT-5.3 and Qwen Coder 8B as baselines.

5.2 Optimization Results

Per-design results. Table 2 summarizes Dr. RTL's optimization results across all 20 designs. Notably, Dr. RTL improves timing on all 20 designs, achieving average WNS and TNS improvements of 21.3% and 16.9%, respectively, while even reducing area by 5.8%. This suggests that the gains mainly come from structurally efficient RTL transformations rather than area-expensive trade-offs. In particular, 9/20 designs achieve Pareto improvements, 8/20 incur only minor area overhead (< 5%), and only 3/20 show noticeable area increase.

Dr. RTL also maintains high reliability throughout optimization, achieving an average SEC pass rate of 86%. Overall, these results show that Dr. RTL delivers consistent timing improvement on realistic human-written RTL through iterative, feedback-driven optimization beyond advanced synthesis.

Comparison with baseline methods. As shown in Table 3, we compare Dr. RTL with single-shot LLMs and representative prior iterative methods [37, 39] on our real-world dataset. Single-shot models are evaluated in one pass, while iterative baselines use the same iteration budget and base model (i.e., Claude Opus) as Dr. RTL. In contrast, prior iterative methods rely mainly on design-level feedback and pre-defined rules, and do not accumulate reusable knowledge from interaction with the synthesis environment.

Table 4: Existing benchmarks fail to capture real RTL optimization. Dr. RTL not only repairs manually degraded RTL, but also consistently improves the original human-written RTL beyond the benchmark's assumed upper bound.

Design	Manual-Degraded Suboptimal RTL			Human-Written RTL (Baseline)			Better Optimized RTL by Dr. RTL		
	WNS	TNS	Area	WNS	TNS	Area	WNS	TNS	Area
FFT	-3.2 (+2%)	-4446 (+0.4%)	160688 (-6%)	-3.14	-4431	170620	-1.73 (-45%)	-1746 (-61%)	110546 (-35%)
Huffman	-2.23 (0%)	-2705 (+0.4%)	73807 (-8%)	-2.23	-2695	80009	-2.21 (-1%)	-2562 (-5%)	71223 (-11%)
VMachine	-0.27 (+145%)	-1.02 (+15%)	20967 (+2%)	-0.11	-0.89	20488	-0.08 (-27%)	-0.38 (-57%)	21989 (+7%)
CNN CPU	Cannot be synthesized using DC								

Dr. RTL consistently outperforms all baselines, achieving the best overall results in WNS, TNS, and area. Among the single-shot baselines, Claude Opus performs better than GPT-5.3, so we adopt it as the backbone model of Dr. RTL. Notably, the iterative baselines [37, 39] also use the same backbone LLM, indicating that the gains mainly come from our agentic optimization framework rather than the underlying model alone.

5.3 Existing Benchmarks Miss Real Optimization

We further evaluate Dr. RTL on a representative RTL optimization benchmark [39], which contains large designs with manually degraded RTL. We include all large designs provided by the benchmark, and note that two of them are not even synthesizable under commercial tools due to syntax issues, further exposing the gap between open-source and industrial EDA evaluation.

We also apply Dr. RTL to this benchmark, as shown in Table 4. Dr. RTL even further improves the original human-written RTL, which existing benchmarks treat as the optimization upper bound. In contrast, prior methods start from manually degraded designs and aim only to recover the original RTL. This shows that the

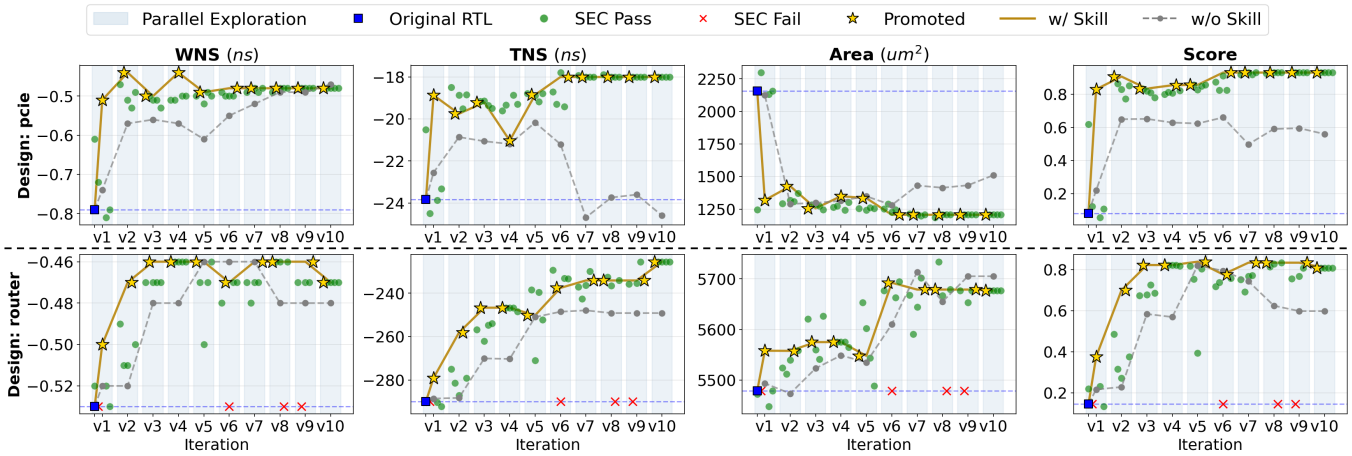


Figure 5: Optimization trajectories of Dr. RTL across iterations of two design examples (router and pcie). WNS, TNS, area, and score improve steadily while preserving functional equivalence. Parallel exploration selects better candidates at each step, and the skill-enhanced variant achieves faster convergence and better PPA than the baseline without skills.

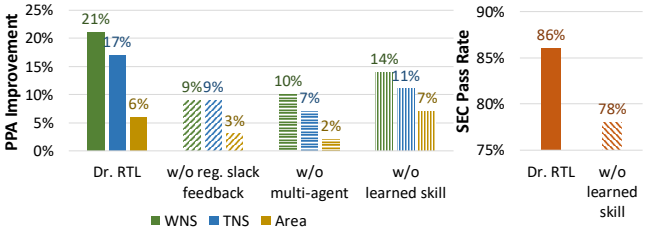


Figure 6: Ablation studies of Dr. RTL on register-level slack feedback, multi-agent design, and learned skill library.

original human-written RTL is not a true upper bound and can still be further optimized under realistic industrial evaluation. More broadly, these results suggest that realistic RTL optimization should be evaluated by whether a method can improve strong human-written RTL beyond synthesis, rather than merely recover it from artificial degradation.

5.4 Optimization Trajectory Visualization

We visualize the optimization trajectories of Dr. RTL in Figure 5. At each iteration, multiple RTL candidates are explored in parallel, and the best SEC-passing design is promoted to the next round. Dr. RTL shows steady improvement in WNS and TNS across iterations, indicating effective convergence. Compared with the variant without the learned skill library, the skill-enhanced version converges faster and reaches better PPA, showing that learned optimization knowledge improves both quality and convergence efficiency.

5.5 Ablation Studies

We conduct ablation studies on the key components of Dr. RTL, as shown in Figure 6. Removing register-level slack feedback causes the largest drop, reducing WNS/TNS improvement from 21%/17% to 9%/9%, highlighting the importance of fine-grained timing information for understanding real critical paths and their root causes. Replacing the multi-agent framework with a single-agent setup reduces performance to 10%/7%, confirming that role decomposition helps structure the optimization process and enables more effective exploration. Disabling the skill library lowers WNS/TNS improvement to 14%/11% and reduces the SEC pass rate from 86% to 78%, indicating that reusable optimization knowledge not only improves search quality but also steers the agent away from ineffective or risky transformations.

5.6 Runtime and Cost Analysis

The runtime of Dr. RTL is dominated by EDA execution, since each candidate requires synthesis and sequential equivalence checking. Because the N candidates in each iteration are evaluated in parallel, wall-clock runtime scales mainly with the number of iterations rather than the total number of candidates, i.e., runtime $\approx K \times T_{\text{EDA}}$, where $K = 10$ in our setup and T_{EDA} is the runtime of one parallel EDA evaluation round. In practice, synthesis dominates T_{EDA}, ranging from minutes to hours depending on design size. Compared with traditional manual RTL optimization, this parallel evaluation enables a substantially more efficient workflow that can run continuously (24×7). In our setup, optimizing all 20 designs required roughly one week of wall-clock time and about $50 in LLM usage. Overall, Dr. RTL provides a favorable cost–performance trade-off while substantially reducing manual effort.

6 DISCUSSION

6.1 Skill Discovery and Interpretability

Figure 7 shows how Dr. RTL converts raw optimization trajectories into reusable RTL optimization knowledge. Hierarchical trajectories record iteration rounds, parallel exploration, and critical-path analyses with root-cause diagnoses, providing structured logs for skill extraction. In our current implementation, this process yields 47 entries in total: 12 high-confidence strategies, 16 medium-confidence strategies, 6 low-confidence strategies, and 13 avoid strategies.

The discovered skills are organized by confidence level according to their empirical success rates under real EDA feedback. High-confidence skills correspond to consistently effective transformations (e.g., logic simplification, fanout management), while medium-confidence skills represent conditionally useful strategies (e.g., restructuring or resource sharing), and low-confidence skills correspond to more aggressive or design-dependent transformations. Dr. RTL also explicitly identifies avoiding strategies that are ineffective, absorbed by synthesis, or violate equivalence.

Each skill is externalized as a pattern–strategy pair with an implementation template, enabling direct reuse. This confidence-aware organization allows Dr. RTL to prioritize reliable transformations while adaptively exploring more complex strategies when beneficial. Overall, Dr. RTL converts raw trajectory experience into structured, interpretable, and reusable knowledge, enabling systematic and adaptive agentic RTL optimization.

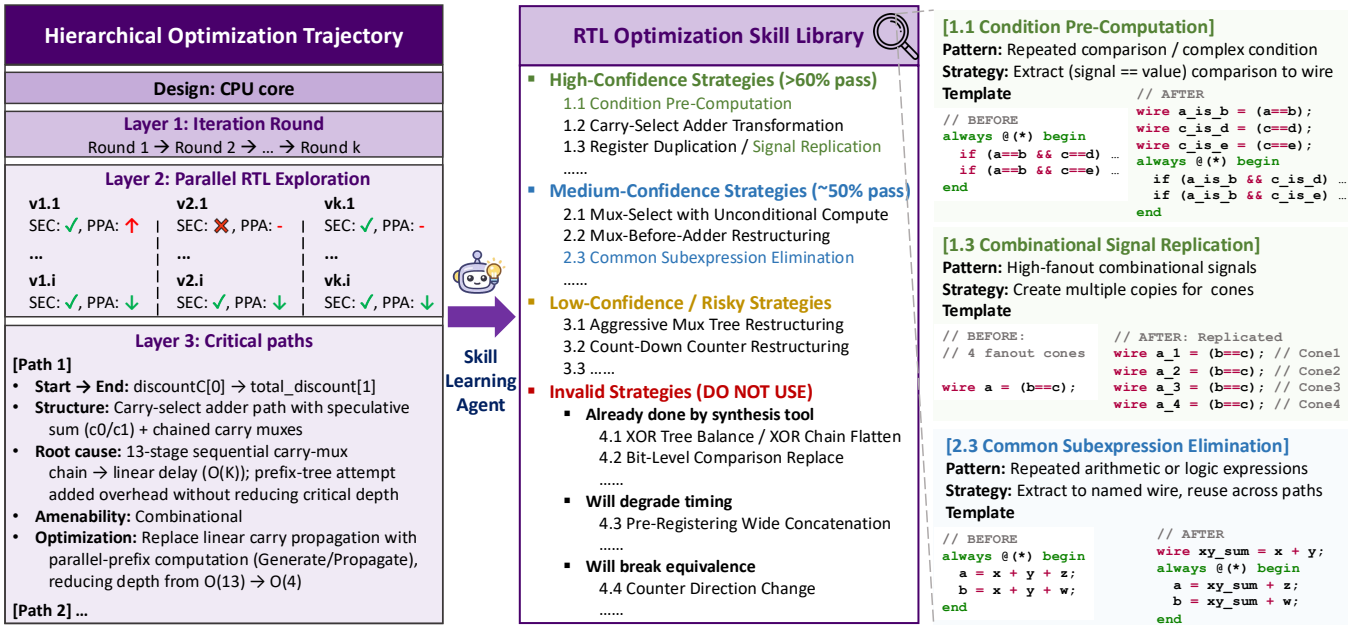


Figure 7: Distilling hierarchical trajectory into skill library. Optimization trajectories are organized across iterations, parallel explorations, and critical paths, enabling discovery of reusable RTL optimization strategies with different confidence levels and associated transformation templates.

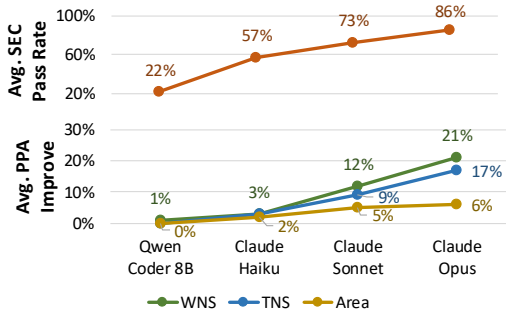


Figure 8: Dr. RTL scales with LLM capability.

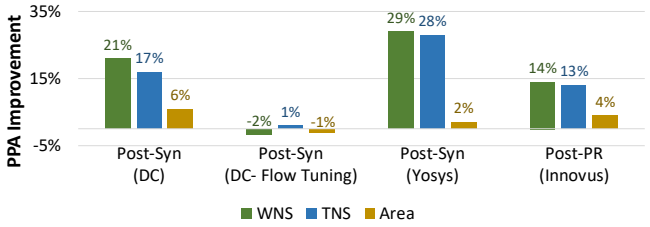


Figure 9: Impact of different EDA tools on Dr. RTL results.

6.2 Dr. RTL Scales with LLM Capability

We study how Dr. RTL scales with different LLM backbones, as shown in Figure 8. Stronger models consistently improve both optimization quality and reliability. Average WNS/TNS improvement increases from 1%/0% with Qwen Coder 8B, to 3%/2% with Claude Haiku, to 12%/9% with Claude Sonnet, and to 21%/17% with Claude Opus. Meanwhile, the average SEC pass rate rises from 22% to 57%, 73%, and 86%. These results suggest that Dr. RTL scales effectively with backbone capability. As smaller models continue to improve, fine-tuned or domain-specialized LLMs are a promising direction for better privacy, customization, and deployment flexibility.

6.3 Impact of EDA Tools

As shown in Figure 9, we evaluate Dr. RTL across multiple EDA tools and design stages, to distinguish true RTL-level optimization from gains that might depend on a particular backend tool or flow setting. Dr. RTL consistently improves timing across all settings, from open-source synthesis to commercial synthesis and post-route evaluation. The gains are largest with Yosys, where the weaker synthesis leaves more room for RTL improvement, while they are smaller but still clear with commercial DC and after place-and-route in Innovus, where timing has already been more aggressively optimized. In contrast, simple DC flow tuning yields only marginal benefit, suggesting that parameter tuning is limited, while substantial further gains still come from improving the RTL itself. Overall, these results show that Dr. RTL effectively generalizes across both tools and design stages.

7 CONCLUSION

We present Dr. RTL, an agentic framework for RTL timing optimization that formulates the task as an iterative, closed-loop process through interaction with industrial EDA tools, achieving consistent timing improvement with minimal area overhead on real-world designs. Beyond performance, our work advocates a shift toward agentic design automation driven by autonomous reasoning, accumulated chip design knowledge, and EDA environment interaction. By coupling realistic evaluation with reusable skill learning, Dr. RTL takes a step toward practical agentic design automation, and paves the way for future systems built on smaller specialized models, richer exploration strategies, and human-in-the-loop refinement.

8 ACKNOWLEDGEMENT

This work is supported by Hong Kong Research Grants Council (RGC) CRF-YCRG C6003-24Y, 16200724, and T46-415/25-R. It is also supported by RGC JLFS-YSF/E-605/26. It was partially conducted by ACCESS – AI Chip Center for Emerging Smart Systems, supported by the InnoHK initiative of the Innovation and Technology Commission of the Hong Kong Special Administrative Region Government.

REFERENCES

[1] Mohammad Akyash, Kimia Azar, and Hadi Kamali. 2025. Rtl++: Graph-enhanced llm for rtl code generation. In *2025 IEEE International Conference on LLM-Aided Design (ICLAD)*. IEEE, 44–50.

[2] Ahmed Allam and Mohamed Shalan. 2024. RTL-Repo: A Benchmark for Evaluating LLMs on Large-Scale RTL Design Projects. *arXiv preprint arXiv:2405.17378* (2024).

[3] Anthropic. 2026. Claude Code: Model Configuration. https://code.claude.com/docs/en/model-config.

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

[5] Che-Ming Chang, Prashanth Vijayaraghavan, Ashutosh Jadhav, Charles Mackin, Hsinyu Tsai, Vandana Mukherjee, and Ehsan Degan. 2026. CODMAS: A Dialectic Multi-Agent Collaborative Framework for Structured RTL Optimization. In *Proceedings of the 19th Conference of the European Chapter of the Association for Computational Linguistics (Volume 5: Industry Track)*. 777–788.

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

[7] Lei Chen et al. 2024. The Dawn of AI-Native EDA: Promises and Challenges of Large Circuit Models. *Springer Science China Information Sciences (SCIS)* (2024).

[8] Luanrong Chen, Renzhi Chen, Xinyu Li, Shanshan Li, Rui Gong, and Lei Wang. 2026. IncreRTL: Traceability-Guided Incremental RTL Generation under Requirement Evolution. *arXiv preprint arXiv:2603.25769* (2026).

[9] Animesh Basak Chowdhury, Marco Romanelli, Benjamin Tan, Ramesh Karri, and Siddharth Garg. 2024. Retrieval-guided reinforcement learning for boolean circuit minimization. In *International Conference on Learning Representations (ICLR)*.

[10] Matthew DeLorenzo, Animesh Basak Chowdhury, Vasudev Gohil, Shailja Thakur, Ramesh Karri, Siddharth Garg, and Jeyavijayan Rajendran. 2024. Make every move count: Llm-based high-quality rtl code generation using mcts. *arXiv preprint arXiv:2402.03289* (2024).

[11] Chenhui Deng, Yun-Da Tsai, Guan-Ting Liu, Zhongzhi Yu, and Haoxing Ren. 2025. Scalertl: Scaling llms with reasoning data and test-time compute for accurate rtl code generation. In *2025 ACM/IEEE 7th Symposium on Machine Learning for CAD (MLCAD)*. IEEE, 1–9.

[12] Chenhui Deng, Zhongzhi Yu, Guan-Ting Liu, Nathaniel Pinckney, and Haoxing Ren. 2026. ACE-RTL: When Agentic Context Evolution Meets RTL-Specialized LLMs. *arXiv preprint arXiv:2602.10218* (2026).

[13] Wenji Fang, Jing Wang, Yao Lu, Shang Liu, Yuchao Wu, Yuzhe Ma, and Zhiyao Xie. 2025. A survey of circuit foundation model: Foundation ai models for vlsi circuit design and eda. *arXiv preprint arXiv:2504.03711* (2025).

[14] Mingzhe Gao, Jieru Zhao, Zhe Lin, Wenchao Ding, Xiaofeng Hou, Yu Feng, Chao Li, and Minyi Guo. 2024. AutoVCoder: A Systematic Framework for Automated Verilog Code Generation using LLMs. In *International Conference on Computer Design (ICCD)*.

[15] Soha Hassoun and Tsutomu Sasao. 2012. *Logic synthesis and verification*. Vol. 654. Springer Science & Business Media.

[16] Zhuolun He, Yuan Pu, Haoyuan Wu, Tairu Qiu, and Bei Yu. 2025. Large language models for eda: Future or mirage? *ACM Transactions on Design Automation of Electronic Systems* 30, 6 (2025), 1–53.

[17] Wei-Po Hsin, Ren-Hao Deng, Yao-Ting Hsieh, En-Ming Huang, and Shih-Hao Hung. 2026. EvolVE: Evolutionary Search for LLM-based Verilog Generation and Optimization. *arXiv preprint arXiv:2601.18067* (2026).

[18] Miao Liu, Liwei Ni, Junfeng Liu, Xingyu Meng, Rui Wang, Xiaoze Lin, Xinhua Lai, Xingquan Li, and Jungang Xu. 2026. A Survey of Machine Learning Approaches in Logic Synthesis. *ACM Transactions on Design Automation of Electronic Systems* 31, 2 (2026), 1–43.

[19] Mingjie Liu, Nathaniel Pinckney, Brucek Khailany, and Haoxing Ren. 2023. VerilogEval: Evaluating Large Language Models for Verilog Code Generation. *arXiv preprint arXiv:2309.07544* (2023).

[20] Shang Liu, Wenji Fang, Yao Lu, Qijun Zhang, Hongce Zhang, and Zhiyao Xie. 2024. RTLCoder: Fully Open-Source and Efficient LLM-Assisted RTL Code Generation Technique. *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems (TCAD)* (2024).

[21] Shang Liu, Yao Lu, Wenji Fang, Mengming Li, and Zhiyao Xie. 2024. OpenLLM-RTL: Open Dataset and Benchmark for LLM-Aided Design RTL Generation. In *International Conference on Computer-Aided Design (ICCAD)*.

[22] Yao Lu, Shang Liu, Qijun Zhang, and Zhiyao Xie. 2024. RTLLM: An Open-Source Benchmark for Design RTL Generation with Large Language Model. In *Asia and South Pacific Design Automation Conference (ASP-DAC)*.

[23] Yao Lu, Shang Liu, Hangan Zhou, Wenji Fang, Qijun Zhang, and Zhiyao Xie. 2026. A New Benchmark for the Appropriate Evaluation of RTL Code Optimization. *arXiv preprint arXiv:2601.01765* (2026).

[24] Kyungjun Min, Kyumin Cho, Junhwan Jang, and Seokhyeong Kang. 2025. Revolution: An evolutionary framework for rtl generation driven by large language models. *arXiv preprint arXiv:2510.21407* (2025).

[25] Jingyu Pan, Guanglei Zhou, Chen-Chia Chang, Isaac Jacobson, Jiang Hu, and Yiran Chen. 2025. A Survey of Research in Large Language Models for Electronic Design Automation. *ACM Transactions on Design Automation of Electronic Systems (TODAES)* (2025).

[26] Zehua Pei, Hui-Ling Zhen, Mingxuan Yuan, Yu Huang, and Bei Yu. 2024. BetterV: Controlled Verilog Generation with Discriminative Guidance. *arXiv preprint arXiv:2402.03375* (2024).

[27] Nathaniel Pinckney, Chenhui Deng, Chia-Tung Ho, Yun-Da Tsai, Mingjie Liu, Wenfei Zhou, Brucek Khailany, and Haoxing Ren. 2025. Comprehensive Verilog design problems: A next-generation benchmark dataset for evaluating large language models and agents on rtl design and verification. *arXiv preprint arXiv:2506.14074* (2025).

[28] Heng Ping, Peiyu Zhang, Zhenkun Wang, Shixuan Li, Anzhe Cheng, Wei Yang, Paul Bogdan, and Shahin Nazarian. 2026. POET: Power-Oriented Evolutionary Tuning for LLM-Based RTL PPA Optimization. *arXiv preprint arXiv:2603.19333* (2026).

[29] Arun Ravindran, Aditya Patra, Vahid Babaey, and Suresh Purini. 2025. Survey and Benchmarking of Large Language Models for RTL Code Generation: Techniques and Open Challenges. (2025).

[30] Humza Sami, Pierre-Emmanuel Gaillardon, Valerio Tenace, et al. 2024. Aivril: Ai-driven rtl generation with verification in-the-loop. *arXiv preprint arXiv:2409.11411* (2024).

[31] Si2. 2018. NanGate 45nm Open Cell Library.

[32] Cadence Design Systems. 2026. Cadence JasperGold Sequential Equivalence Checking App. https://www.cadence.com/en_US/home/tools/system-design-and-verification/formal-and-static-verification/jasper-verification-platform/jaspergold-sequential-equivalence-checking-app.html.

[33] Kimia Tasnia, Alexander Garcia, Tasnuva Farheen, and Sazadur Rahman. 2025. Veriopt: Ppa-aware high-quality verilog generation via multi-role llms. In *2025 IEEE/ACM International Conference On Computer Aided Design (ICCAD)*. IEEE, 1–9.

[34] Kiran Thorat, Jiahui Zhao, Yaotian Liu, Amit Hasan, Hongwu Peng, Xi Xie, Bin Lei, and Caiwen Ding. 2025. LLM-VeriPPA: Power, Performance, and Area Optimization aware Verilog Code Generation with Large Language Models. In *2025 ACM/IEEE 7th Symposium on Machine Learning for CAD (MLCAD)*. IEEE, 1–7.

[35] Kiran Thorat, Jiahui Zhao, Yaotian Liu, Hongwu Peng, Xi Xie, Bin Lei, Jeff Zhang, and Caiwen Ding. 2023. Advanced Large Language Model (LLM)-Driven Verilog Development: Enhancing Power, Performance, and Area Optimization in Code Synthesis. *arXiv preprint arXiv:2312.01022* (2023).

[36] Frank Vahid. 2010. *Digital design with RTL design, VHDL, and Verilog*. John Wiley & Sons.

[37] Yiting Wang, Wanghao Ye, Ping Guo, Yexiao He, Ziyao Wang, Bowei Tian, Shwai He, Guoheng Sun, Zheyu Shen, Sihan Chen, et al. 2025. Symrtlo: Enhancing rtl code optimization with llms and neuron-inspired symbolic reasoning. In *Advances in Neural Information Processing Systems (NeurIPS)*.

[38] Zhihao Xu, Bixin Li, and Lulu Wang. 2025. Rethinking LLM-Based RTL Code Optimization Via Timing Logic Metamorphosis. *arXiv preprint arXiv:2507.16808* (2025).

[39] Xufeng Yao, Yiwen Wang, Xing Li, Yingzhao Lian, Ran Chen, Lei Chen, Mingxuan Yuan, Hong Xu, and Bei Yu. 2024. Rtlrewriter: Methodologies for large models aided rtl code optimization. In *Proceedings of IEEE/ACM International Conference on Computer-Aided Design (ICCAD)*.

[40] Jiaqi Yin, Zhan Song, Chen Chen, Qihao Hu, and Cunxi Yu. 2025. Boole: Exact symbolic reasoning via boolean equality saturation. In *2025 62nd ACM/IEEE Design Automation Conference (DAC)*. IEEE, 1–7.

[41] Zhongzhi Yu, Mingjie Liu, Michael Zimmer, Yingyan Celine, Yong Liu, and Haoxing Ren. 2025. Spec2rtl-agent: Automated hardware code generation from complex specifications using llm agent systems. In *2025 IEEE International Conference on LLM-Aided Design (ICLAD)*. IEEE, 37–43.

[42] Zelin Zang, Yuhang Song, Aili Wang, Bingo Wing-Kuen Ling, Qi Sun, Zhen Lei, Fuji Yang, Cheng Zhuo, and Jiebo Luo. 2025. The Dawn of Agentic EDA: A Survey of Autonomous Digital Chip Design. *arXiv preprint arXiv:2512.23189* (2025).

[43] Xinyu Zhang, Zhiteng Chao, Yonghao Wang, Bin Sun, Tianyun Ma, Tianmeng Yang, Jianan Mu, Jing Justin Ye, and Huawei Li. 2026. RTLSeek: Boosting the LLM-Based RTL Generation with Multi-Stage Diversity-Oriented Reinforcement Learning. *arXiv preprint arXiv:2603.27630* (2026).

[44] Yang Zhao, Di Huang, Chongxiao Li, Pengwei Jin, Ziyuan Nan, Tianyun Ma, Lei Qi, Yansong Pan, Zhenxing Zhang, Rui Zhang, et al. 2024. CodeV: Empowering LLMs for Verilog Generation through Multi-Level Summarization. *arXiv preprint arXiv:2407.10424* (2024).

[45] Yujie Zhao, Hejia Zhang, Hanxian Huang, Zhongming Yu, and Jishen Zhao. 2025. Mage: A multi-agent engine for automated rtl code generation. In *2025 62nd ACM/IEEE Design Automation Conference (DAC)*. IEEE, 1–7.

[46] Matthew M Ziegler, Hung-Yi Liu, George Gristede, Bruce Owens, Ricardo Nigaglioni, and Luca P Carloni. 2016. A synthesis-parameter tuning system for autonomous design-space exploration. In *2016 Design, Automation & Test in Europe Conference & Exhibition (DATE)*. IEEE, 1148–1151.