Cedar: Learning to Optimize RTL via Accumulating Verified Rewrite Rules

Niansong Zhang
Cornell University
Ithaca, New York, USA
nz264@cornell.edu

Chenhui Deng
NVIDIA
Santa Clara, California, USA
cdeng@nvidia.com

Johannes Maximilian Kuehn
NVIDIA
Santa Clara, California, USA
jkuhn@nvidia.com

Chia-Tung Ho
NVIDIA
Santa Clara, California, USA
chiatungh@nvidia.com

Cunxi Yu
University of Maryland, College Park
College Park, Maryland, USA
cunxiyu@umd.edu

Zhiru Zhang
Cornell University
Ithaca, New York, USA
zhiruz@cornell.edu

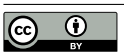
Brucek Khailany
NVIDIA
Austin, Texas, USA
bkhailany@nvidia.com

Abstract

Pre-synthesis RTL optimization can unlock improvements that downstream logic synthesis alone cannot reach, yet existing e-graph-based methods are confined to small datapaths, depend on hand-crafted rewrite rules, and do not jointly optimize power, delay, and area. We present Cedar, which extends equality saturation to RTL an order of magnitude beyond prior e-graph work through an LLM-driven optimization loop that *accumulates formally verified rewrite rules across designs*: rules discovered while optimizing one circuit are reused on future, unseen ones. Extended e-graph operators and direct semantic rewrites enable Cedar to handle complex RTL (multiplexers, FSMs, decoders, and heterogeneous datapaths), and a power-aware extraction cost model enables joint PPA optimization. Evaluated on 88 designs (44 RTLLM v2.0 benchmarks, 40 RTL-OPT benchmarks, the Snitch IPU and multiplier from PULP MemPool, and two NVDLA MAC modules) in the ASAP7 7 nm PDK, Cedar achieves geometric-mean reductions of 19.8% area and 18.9% power on RTLLM, and 32.3% area and 22.8% power on RTL-OPT, outperforming direct LLM editing (11.9% area) and evolutionary LLM search (27.3% area). On the large designs, Cedar reduces Snitch IPU post-PnR area / power by 23.9% / 46.1%, Snitch multiplier area by 47.1%, and NVDLA MAC power by up to 13.7%.

CCS Concepts

• **Hardware → High-level and register-transfer level synthesis**; **Hardware description languages and compilation**; • **Computing methodologies → Multi-agent systems**; **Intelligent agents**.



This work is licensed under a Creative Commons Attribution 4.0 International License.
ICCAD '26, San Jose, CA, USA
© 2026 Copyright held by the owner/author(s).
ACM ISBN 979-8-4007-2873-0/2026/11
https://doi.org/10.1145/3831252.3834087

Keywords

RTL optimization, equality saturation, e-graphs, large language models, rewrite rules, power optimization

ACM Reference Format:
Niansong Zhang, Chenhui Deng, Johannes Maximilian Kuehn, Chia-Tung Ho, Cunxi Yu, Zhiru Zhang, and Brucek Khailany. 2026. Cedar: Learning to Optimize RTL via Accumulating Verified Rewrite Rules. In *IEEE/ACM International Conference on Computer-Aided Design (ICCAD '26), November 08–12, 2026, San Jose, CA, USA.* ACM, New York, NY, USA, 9 pages. https://doi.org/10.1145/3831252.3834087

1 Introduction

Consider optimizing the Integer Processing Unit (IPU) inside the Snitch core [30], replicated 256 times in the PULP MemPool cluster [17]. Within the IPU, the integer execute core (the ALU, shifter, comparator, and result multiplexer datapath) is a ∼14K-gate block whose rewriting we target in this work; the full IPU, which additionally includes decoder and register-file logic, is larger. A 24% area reduction on this execute core saves ∼1.08 mm^2 of silicon cluster-wide; a 46% power reduction saves ∼68.6 mW. These are meaningful gains, yet no automated RTL optimizer has previously handled a design of this scale. Equality saturation, the most principled approach to RTL rewriting [28], has been limited to small, pure-arithmetic datapaths under 1K gates: FIR filters, polynomial evaluators, constant multipliers [7, 31]. The IPU execute core, with its interleaved MAC unit, division logic, comparison branches, and result multiplexer tree, lies entirely outside this scope.

The fundamental obstacle is not the e-graph itself, but three surrounding bottlenecks. First, real RTL designs are too large and heterogeneous for a single e-graph: mixing arithmetic, control, and multiplexing logic in one expression graph causes combinatorial blow-up. Second, the rewrite rules that drive equality saturation are hand-crafted for narrow circuit families [6, 7], and rules designed for one family (e.g., arithmetic identities) fail to optimize another (e.g., mux cascades). Third, prior e-graph RTL work targets primarily delay and area [7, 31]; recent work incorporates power on narrow

Table 1: Comparison of Cedar with LLM-based RTL optimization methods and e-graph-based approaches. ✓ = supported, ✗ = not supported. †POET uses differential-testing testbenches, not formal equivalence. Opus 4.6 uses the same LLM as Cedar's agent but without verified rules or eqsat. Largest design in gates.

		Optimization			Verification		Scalability	
Method	Approach	Joint PPA	Cross-Design Transfer	Eqsat Search	Formal Verif.	Func. Correct.	Beyond Datapath	Largest Design
ChatGPT-4 [14]	Direct LLM	✗	✗	✗	✗	~75%	✓	<1K
VeriOpt [18]	Multi-role LLM	✓	✗	✗	✗	86%	✓	<1K
CODMAS [1]	Dialectic agents	✓	✗	✗	✗	~70%	✓	<1K
POET [16]	Evolutionary LLM	✓	✗	✗	Sim†	100%	✓	<1K
RTL-OPT (Opus 4.6)	Direct LLM	✗	✗	✗	✗	~87%	✓	<1K
ROVER [7, 9]	Fixed rules + eqsat	✓	✗	✓	✓	100%	✗	<1K
ASPEN [31]	LLM rules + eqsat	✗	✗	✓	✓	100%	✗	<1K
Cedar	**Acc. rules + eqsat**	✓	✓	✓	✓	**100%**	✓	**14K**

arithmetic datapaths [9], but joint optimization of power, delay, and area on full RTL designs with a unified Pareto front remains open.

We propose Cedar[1], an LLM agent that addresses all three bottlenecks simultaneously. The agent *proposes* new rewrite rules tailored to each design's structure, *accumulates* formally verified rules into a shared pool that grows stronger with every design it optimizes, and *applies* both e-graph exploration and direct semantic rewrites to handle designs beyond the reach of the CIRCT pipeline.

The evolving rule pool is the central mechanism. Starting from 14 algebraic identities, the pool grows as the LLM proposes candidate rules, Z3 verifies their correctness, and verified rules are stored for future reuse. After optimizing 40+ designs, the experiment-freeze snapshot contains 175 Z3-verified rules within 207 total candidates, spanning arithmetic decomposition, multiplexer restructuring, Boolean simplification, and comparison optimization. Rules discovered on small benchmarks transfer directly to larger, unseen designs: MAC optimization rules from 8-bit multiplier benchmarks enable the `shared_mac` variant on the 14K-gate Snitch IPU without any design-specific engineering.

This work makes the following contributions:

- **Scalable RTL optimization an order of magnitude beyond prior work.** CIRCT/MLIR/egglog integration, extended e-graph operators let Cedar handle 88 designs across 10 categories, including the 14K-gate Snitch IPU and NVDLA CACC leaf modules.
- **An evolving, verification-gated rule pool with cross-design transfer.** The evaluation snapshot contains 207 rules and candidates, with accepted rewrites gated by Z3 verification. On 9 unseen RTL-OPT designs, the frozen evaluation pool achieves 29.1% geometric-mean area reduction, compared with 31.7% for per-design from-scratch optimization, and rules from 8-bit multipliers directly enable Snitch IPU optimizations.
- **Joint PPA optimization with power-aware extraction.** A three-objective cost model with switching-activity-based power estimation discovers Pareto-optimal points, yielding 46.1% power and 23.9% post-PnR area reduction on the Snitch IPU.
- **Empirical analysis of why local rewrites help.** Classifying 10 representative rules against a commercial RTL synthesis tool, 5 target *structural fidelity* (the tool applies them in isolation but not in larger designs), 4 are *genuinely novel* transformations, and 1 is a context-dependent rewrite harmful in isolation but beneficial through equality saturation, demonstrating the value of global search over greedy application.

Cedar rewrites combinational logic while preserving the register placement, state encoding, and cycle-level I/O behavior. Sequential transformations such as retiming, FSM re-encoding are orthogonal and complementary, but outside the scope of this work.

2 Background and Related Work

2.1 Equality Saturation and E-Graphs

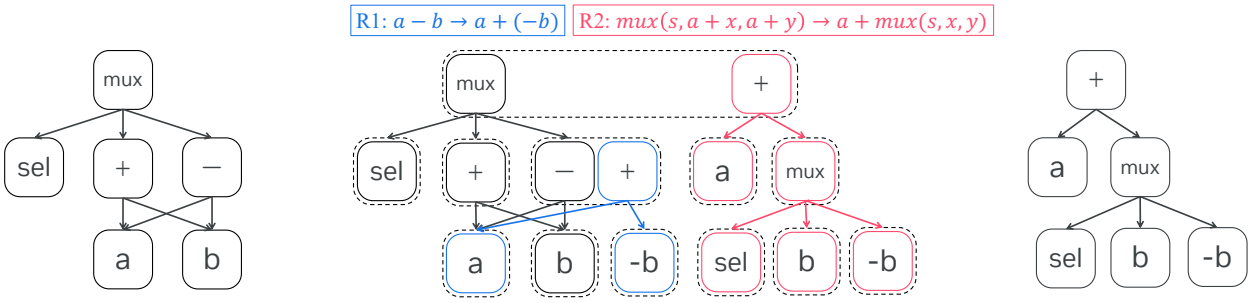
An e-graph compactly represents equivalent expressions by grouping them into *e-classes* [28]. Each e-class contains one or more *e-nodes* (operators whose children are other e-classes). Rewrite rules apply non-destructively: matching a rule adds the rewritten expression to the same e-class without removing the original. This process, *equality saturation*, continues until a fixed point or timeout, after which *extraction* selects a minimum-cost expression. Figure 1 illustrates this on a running example where two interacting rules produce an optimized single-adder design; notably, the first rule is harmful in isolation but enables the second, motivating global search over greedy application. The `egg` library [28] and `egglog` [32] provide efficient implementations, applied in linear algebra [26], signal processing [24], and logic synthesis [2, 3, 29].

2.2 E-Graph-Based RTL Optimization

ROVER [7] first applied e-graphs to RTL optimization, defining bitwidth-aware rewrite rules for arithmetic datapaths. Subsequent work added constraint-aware transformations [6, 8] and power optimization [9]. Ustun et al. [23] analyzed equality saturation as a pathway to Pareto optimality, Cheng et al. [4] applied it to HLS, and E-syn [3] introduced technology-aware cost functions.

Two limitations pervade this body of work. First, all systems rely on manually engineered rules: maintaining an effective set requires deep domain expertise, and rules designed for one circuit family rarely transfer to another. Second, adding too many rules causes e-graph blow-up, creating a tension between coverage and tractability. ASPEN [31] uses LLMs to propose and select rules with real PPA feedback, achieving 16.5% area and 6.7% delay improvement

[1]Cedar is open-source and available at https://github.com/cornell-zhang/cedar.



(a) Input design: `y=sel?(a-b):(a+b)` (b) Saturated e-graph after applying R1, R2 (c) Extracted design: -11.5% area, -26.3% power

Figure 1: Equality saturation on `y = sel ? (a−b) : (a+b)`. (a) Original design with separate subtractor and adder. (b) R1 (blue) rewrites subtraction as addition of negation (+60% area alone); R2 (red) rewrites multiplexed addition into a single adder with multiplexed input. (c) Extraction selects the single-adder design: −11.5% area, −26.3% power (−30.6% area, −29.8% power on an 8-bit ALU).

over ROVER on seven datapath benchmarks. However, ASPEN was limited to small datapaths, did not address power, and discarded rules after each design.

2.3 LLMs for RTL Design

LLMs have been applied to RTL generation [10–12, 14, 19, 27] and agent-based workflows [20, 21, 33]. Recent work combines LLMs with symbolic reasoning [25] or synthesis feedback [22].

LLM-Based PPA Optimization. Concurrent work applies LLMs directly to PPA optimization. POET [16] combines evolutionary search with LLM-driven mutation on 40 RTL-OPT designs, achieving 100% correctness via differential testing. CODMAS [1] uses a dialectic multi-agent framework for pipelining and clock gating on 120 designs. VeriOpt [18] employs multi-role prompting on 29 RTLLM designs with an 86% functional success rate. All three directly edit RTL, so correctness depends on simulation coverage (Table 1). Cedar instead uses LLMs to propose rules that are formally verified via Z3 before driving equality saturation, providing provable equivalence.

3 Methodology

Figure 2 summarizes the two-stage workflow. First, the LLM directly rewrites RTL using synthesis feedback, producing verified variants and evidence for rule discovery. It then proposes rules from PPA-impact analysis, formally verifies and accumulates them, selects a design-specific subset, and runs equality saturation. Extraction and synthesis/power feedback update the cost model for the next iteration.

Running Example (Figure 1). Consider `y = sel ? (a-b) : (a+b)`. The 14 initial rules find no rewrite. The LLM proposes R1: $a - b \rightarrow a + (-b)$ and R2: $\text{mux}(s, a+x, a+y) \rightarrow a + \text{mux}(s, x, y)$; Z3 proves both valid over 32-bit bitvectors. R1 alone *increases* area by 60%, but enables R2, which merges both paths into a single adder for −11.5% area and −26.3% power. Neither commercial nor open-source synthesis tools perform this mux-arithmetic interchange. The Snitch IPU's divider stages later reuse both rules from the pool, showing how *rules accumulate, transfer, and compound.*

3.1 Scaling Beyond Datapath Optimization

Unlike ROVER [7] and ASPEN [31], whose custom translators are limited to arithmetic datapaths, Cedar combines CIRCT's Verilog/MLIR infrastructure [13] with egglog [32]. We extend the translation with first-class multiplexer, comparison, and bit-manipulation nodes, supporting FSMs, decoders, multiplexer trees, and heterogeneous datapaths an order of magnitude larger than prior e-graph RTL targets.

Direct Semantic Rewrites. In addition to e-graph exploration, the agent generates pool-guided RTL variants using architectural transformations such as encoding changes, resource sharing, and logic restructuring. The two paths are complementary: eqsat systematically explores fine-grained algebraic equivalences, while direct rewrites handle coarse changes that cannot be expressed as local rules. The best result from either path enters the Pareto frontier after simulation or functional-equivalence checking (Section 3.4).

3.2 Evolving Rule Pool

Prior systems either fix the rule set at design time [7] or generate rules per-design and discard them [31]. Cedar instead maintains a shared, growing pool of formally verified rules.

Rule Proposal. For each design, the LLM agent analyzes the e-graph after initial saturation and proposes candidate rules targeting observed patterns. The agent receives a serialized egglog s-expression of the e-graph (truncated to fit within the context window) along with the current PPA metrics and the design's RTL source, then proposes `lhs` ⟶ `rhs` rewrite rules in egglog syntax. Rules span four categories: *arithmetic decomposition* (shift-add chains for constant multiplication), *multiplexer restructuring* (cascaded-to-tree conversion, shared-branch merging), *Boolean simplification* (De Morgan's laws, redundant logic elimination), and *comparison optimization* (deriving $a > b$ from $\neg(a < b) \wedge \neg(a = b)$).

Rule Selection. Given the pool, current e-graph, and historical per-rule PPA impact, the agent (1) retains rules whose left-hand sides occur in the graph, (2) prioritizes rules that previously improved area or power by at least 1% on similar designs, (3) confirms

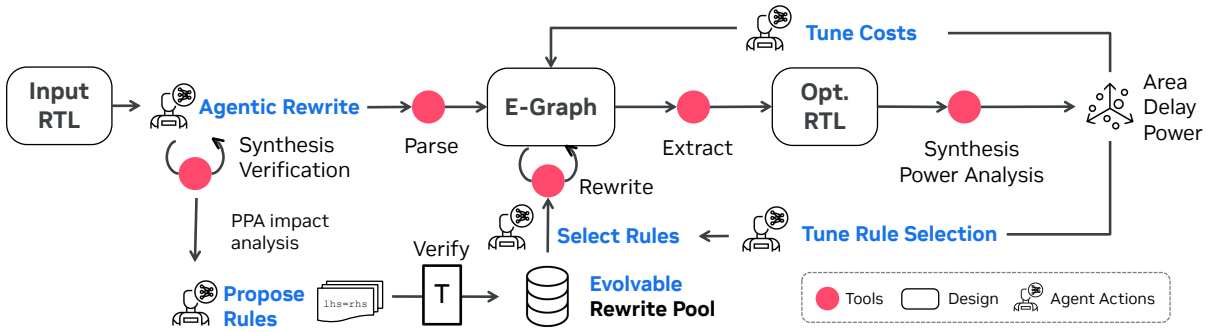


Figure 2: Cedar workflow. Stage 1: the agent directly rewrites RTL and runs synthesis to discover PPA-improving changes; correctness is checked by simulation or equivalence checking. Stage 2: the agent proposes rewrite rules from PPA analysis, verifies them with Z3, and adds them to an evolvable pool. Selected rules drive equality saturation; extraction and synthesis feedback tune costs for the next iteration.

that candidates produce new e-nodes, and (4) groups complementary rules. For example, on `addr_calcu`, subtraction and addition factoring rules transform `mux(c,a−x+b,a−y+b)` by first sharing the subtraction and then the common `+b`; addition commutativity enables matching when operand order differs. Together they reduce area by 36.8%, whereas either factoring rule alone yields only 0.3%. Synthesis feedback guides the combinations explored in later iterations.

Formal Verification. Each candidate rule is verified by encoding $lhs \neq rhs$ as a quantifier-free bitvector formula and checking unsatisfiability with Z3. If the formula is UNSAT, the rule is valid for all inputs at the specified width. Only verified rules enter the pool. Rules are verified per concrete bitwidth (e.g., 8-bit, 16-bit, 32-bit) with a 60-second solver timeout per rule. Each rule is stored with width guards that prevent misapplication across bitwidths; when a rule is needed at a new width, the agent re-verifies it before use.

Pool Accumulation. Verified rules are stored in a shared pool available to every design. Starting from 14 algebraic identities, the pool has grown to 175 rules across 40+ designs, including arithmetic decomposition, multiplexer restructuring, Boolean simplification, and comparison optimization.

3.3 Joint PPA Cost Model

ASPEN [31] used synthesis feedback for area-delay optimization only. Cedar adds power as a first-class objective.

Power Estimation. Gate-level simulation supplies switching activity at the common per-design clock frequency (Section 4.3). A netlist-to-e-node map apportions shared gate power among contributing nodes. Although approximate, this estimate compares extraction alternatives within the same e-class.

Cost Adjustment. The e-node cost function combines three objectives:

$$\text{cost}(n) = \alpha \cdot \text{area}(n) + \beta \cdot \text{delay}(n) + \gamma \cdot \text{power}(n) \qquad (1)$$

The exploration loop varies α, β, γ to discover different Pareto points. The PPA feedback agent increases costs for nodes that contribute disproportionately to power or area, biasing subsequent extractions toward alternative implementations. This is how Cedar discovers the Snitch `gated_mac` variant (21.9% power reduction): area-only extraction misses it because the gated implementation uses slightly more area.

Algorithm 1 Cedar: Two-level optimization with evolving rule pool and joint PPA feedback

Require: Design D, rule pool $\mathcal{P}$, initial rules $\mathcal{R}_0$
Ensure: Pareto frontier $\mathcal{F}$, updated pool $\mathcal{P}'$

$\mathcal{R}_{\text{new}} \leftarrow$ LLM_ProposeRules$(D, \mathcal{R}_0, \mathcal{P})$
$\mathcal{R}_{\text{verified}} \leftarrow$ Z3_Verify$(\mathcal{R}_{\text{new}})$
$\mathcal{P}' \leftarrow \mathcal{P} \cup \mathcal{R}_{\text{verified}}$ ▷ Evolving pool
for each rule selection iteration **do**
  $\mathcal{R} \leftarrow$ LLM_SelectRules$(\mathcal{P}', D, \text{history})$
  $G \leftarrow$ Egglog_Saturate$(D, \mathcal{R})$
  costs $\leftarrow$ InitCosts(α, β, γ) ▷ PPA weights
  for each PPA feedback iteration **do**
    expr $\leftarrow$ Extract(G, costs)
    $V \leftarrow$ TranslateToVerilog(expr)
    (area, delay, power) $\leftarrow$ Synthesize(V)
    if Simulate(V) passes **then**
      Update $\mathcal{F}$ if non-dominated
    end if
    for each partition p in netlist **do**
      $(inc, dec) \leftarrow$ LLM_Trace(V, p, ppa)
      Adjust costs for critical e-nodes
    end for
  end for
end for
return $\mathcal{F}, \mathcal{P}'$

3.4 Verification

Correctness is checked at two shipped-evidence levels: Z3 bitvector proofs for accepted rules and cycle-accurate simulation or a functional-equivalence miter against the original RTL. E-graph congruence preserves the soundness of verified rules under composition; direct semantic rewrites are admitted only after the end-to-end functional check passes.

4 Experimental Setup

4.1 Benchmarks

We evaluate on three suites of increasing complexity.

RTLLM v2.0 Suite (44 designs). Drawn from the RTLLM v2.0 benchmark [12, 14]: 44 designs from the 50-design suite (excluding 4 non-synthesizable or corrupt designs and 2 duplicates with RTL-OPT). Categories include arithmetic (`adder_32bit`, `alu`), FSMs and controllers (`traffic_light`, `sequence_detector`), data converters and memory (`accu`, `RAM`), multipliers (`multi_pipe_8bit`), and frequency dividers. Designs range from 12 gates to 3.6K gates and include sequential logic that stresses the CIRCT extraction pipeline.

RTL-OPT Suite (40 designs). The 40-design RTL-OPT benchmark used by POET [16], spanning eight categories: adders (4), ALUs (2), comparators (5), decoders (2), dividers (4), FSMs (2), multipliers (6), and multiplexers/control (15). Unlike prior e-graph benchmarks [7, 31], this suite includes control flow, mixed arithmetic-control logic, and complex multiplexing.

Large Design Case Studies (3 designs). Two modules from the PULP MemPool Snitch cluster [17, 30]: the integer execute core of the Snitch IPU (∼590 LOC, ∼14K cells) and the Snitch multiplier (116 LOC). MemPool replicates each Snitch core 256 to 1024×, amplifying per-core gains. The full convolution accumulator (CACC) subsystem from NVIDIA's NVDLA [15]: a 25-file, 31K-LOC hierarchical design with 13 core modules plus RAM macros, representing a different application domain (DNN acceleration) and testing Cedar's ability to optimize multi-file RTL projects rather than isolated modules.

4.2 Baselines

Four baselines form an ablation ladder, each adding one component:

(1) **CAD Tool Baseline**: unoptimized design, direct synthesis.
(2) **Agent Rewrite**: LLM directly edits RTL with simulation validation. No rule pool, no e-graph, no workflow.
(3) **Agent + Rule Pool**: LLM applies pool rules by direct RTL editing, without equality saturation or structured workflow.
(4) **Agent + Rule Pool + Eqsat**: LLM uses both pool and eqsat tools, but without systematic rule selection, PPA feedback, or iterative cost adjustment.

4.3 Synthesis and Power Estimation

All designs are synthesized with a state-of-the-art commercial RTL synthesis tool using ASAP7 7 nm [5] at high optimization effort. Clock frequency per design is set by shmoo analysis: we sweep from low frequency upward and select the knee, defined as the lowest frequency at which area increases by more than 5% per 100 MHz step (evaluated automatically per design; per-design frequencies range from 0.02 to 14 GHz).

Power estimation uses Synopsys PrimeTime with back-annotated switching activity. The flow proceeds in four stages: (1) logic synthesis produces a gate-level netlist; (2) place-and-route extracts post-PnR parasitics (SPEF) and the annotated netlist; (3) gate-level simulation with SDF back-annotation runs each design's functional testbench, dumping per-net switching activity (SAIF); (4) PrimeTime reads the post-PnR netlist, SPEF parasitics, and SAIF activity to produce the final power report. This SAIF-based flow captures realistic switching activity from testbench stimulus rather than relying on statistical toggle-rate assumptions. All configurations for a given design are compared at the same clock frequency and activity source, ensuring fair comparison.

Table 2: Representative RTLLM results on 44 designs (ASAP7 7 nm, shmoo frequency per design).

		CAD Baseline			Cedar		
Cat.	**Design**	**A (μm^2)**	**D (ns)**	**P (μW)**	ΔA (%)	ΔD (%)	ΔP (%)
FSM	`traffic_light`	162.1	0.21	18.2	−7.6%	0.0%	−5.4%
	`sequence_det.`	56.2	0.13	6.7	−43.6%	−33.9%	−38.4%
	`up_down_counter`	239.8	0.41	26.7	−33.0%	−3.7%	−26.8%
	Geo. mean (7)	—	—	—	−21.5%	−13.0%	−27.8%
Arith.	`adder_32bit`	650.4	0.51	148.6	−77.3%	−57.9%	−11.7%
	`adder_8bit`	50.4	0.35	46.8	−24.5%	−42.3%	−6.5%
	`alu`	2926.3	0.95	304.8	−35.8%	−0.7%	−30.7%
	`div_16bit`	767.3	3.09	127.7	−64.0%	−61.3%	−86.6%
	`multi_pipe_8bit`	965.3	0.38	103.2	−11.8%	−1.4%	−20.3%
	`barrel_shifter`	67.9	0.07	44.6	−54.3%	0.0%	−2.1%
	Geo. mean (17)	—	—	—	−26.7%	−19.8%	−26.7%
Data	`RAM`	777.5	0.14	100.3	−29.6%	0.0%	−16.3%
	`width_8to16`	199.7	0.16	28.8	0.0%	−12.9%	−0.1%
	`adder_pipe_64b`	2788.9	0.47	106.9	−62.5%	0.0%	−74.9%
	Geo. mean (7)	—	—	—	−11.5%	−9.6%	−10.7%
Misc.	`pulse_detect`	19.8	0.07	5.0	−14.1%	−30.5%	−5.8%
	`asyn_fifo`	1344.6	0.10	74.4	0.0%	−50.3%	0.0%
	`pe`	3294.8	1.31	286.5	0.0%	−14.4%	−19.2%
	Geo. mean (13)	—	—	—	−13.3%	−22.9%	−6.4%
Geo. Mean (44)		—	—	—	−19.8%	−18.2%	−18.9%

Area in μm^2, delay in ns, power in μW (PrimeTime SAIF). Δ: best % change vs. baseline (negative = improvement). Per-category means over all designs in that category (counts in parentheses).

4.4 LLM and Eqsat Configuration

Rule proposal uses Claude Opus 4.6 (stronger reasoning for Z3 proofs); rule selection and PPA feedback use Claude Sonnet 4.6. Equality saturation uses `egglog` [32] and greedy bottom-up extraction. Each design explores ≥3 cost configurations and ≥3 rule combinations across ≥10 iterations.

5 Results

5.1 RTLLM v2.0 Benchmark Results

Table 2 presents representative results on 44 RTLLM v2.0 designs [12, 14] (ASAP7 7 nm, shmoo frequency per design); we present representative designs here. Of the 50 RTLLM v2.0 designs, we exclude 6: `adder_16bit`, `float_multi`, `clkgenerator`, and `ring_counter` are non-synthesizable, simulation-only, or contain corrupt simulation data; `fsm` and `comparator_4bit` overlap with the RTL-OPT suite and are evaluated there.

Eqsat Results. 42 designs achieve PPA improvement in at least one metric. The strongest gains are on arithmetic and divider designs: `adder_32bit` (77.3% area / 11.7% power against the default hierarchical baseline; 34.5% area against the strongest flattening plus datapath-optimization baseline), `div_16bit` (64.0% area / 61.3% delay / 86.6% power), `alu` (35.8% area), `barrel_shifter` (54.3% area against the default hierarchical baseline; approximately 5% under full flattening), and `sequence_detector` (43.6% area / 33.9% delay / 38.4% power). Designs mixing sequential control with arithmetic, previously outside the reach of e-graph RTL optimizers, now show real improvements: `freq_divbyodd` yields 81.3% delay reduction via a one-hot encoding change that no prior rule library targeted; `RAM` achieves 29.6% area and 16.3% power reduction; `asyn_fifo`

Table 3: Ablation over 16 RTLLM/RTL-OPT designs. Entries are mean changes from the commercial CAD baseline; negative is improvement.

Configuration	ΔArea	ΔDelay	ΔPower
CAD Tool Baseline	0.0%	0.0%	0.0%
Agent Rewrite (direct edit)	−20.7%	−0.6%	−7.3%
Agent + Rule Pool (no search)	−11.4%	−4.5%	−1.1%
Eqsat + Rule Pool (no search)	−11.5%	−4.4%	−1.0%
Cedar w/o direct rewrites	−13.3%	0.0%	−0.1%
Full Cedar (workflow + PPA feedback)	−19.8%	−6.8%	−10.0%

achieves 50.3% delay reduction. Reported optimized points are gated by the shipped rule-verification, simulation, and functional-equivalence evidence.

Overall. Across all 44 designs, Cedar achieves 19.8% area, 18.2% delay, and 18.9% power reduction (geometric mean). Only 2 designs (`edge_detect`, `right_shifter`) remain at 0%, as they are already in canonical form (a 5-gate edge detector and an 8-flop SIPO chain).

5.2 Case Study: Large Designs from Two Domains

To evaluate Cedar beyond small benchmarks, we target three large designs from two different domains: two modules from the PULP MemPool Snitch cluster [17, 30] and the full convolution accumulator (CACC) subsystem from NVIDIA's Deep Learning Accelerator (NVDLA) [15]. All power is measured with PrimeTime SAIF.

Snitch IPU (590 LOC, ~14K gates). The integer execute core (ALU, shifter, comparator, result multiplexer), reported post place-and-route (Innovus cell area, Innovus post-route critical path, PrimeTime SAIF power), measures 17,600 μm^2, 1.37 ns, 581.6 μW at the baseline. Cedar achieves 23.9% post-PnR area reduction (13,400 μm^2) and 46.1% power reduction (313.5 μW), with timing closure preserved (TNS = 0 across all iterations). A separate variant improves the post-PnR critical path by 4.1% (1.32 ns). At 256 cores, savings translate to ~1.08 mm^2 area and ~68.6 mW cluster-wide.

Snitch Multiplier (116 LOC).. The `snitch_multiplier`, a sibling module of the IPU in the Snitch cluster, synthesizes to 49,023 μm^2 and 5.0 ns delay. Cedar achieves 47.1% area reduction and 16.6% delay improvement via a direct rewrite that factors signed and unsigned multiplication paths through a shared unsigned core, at an 11.9% power cost. This is the largest single-design area reduction in our evaluation.

NVDLA CACC (31K LOC, 25 files). To test scalability beyond single modules, we run Cedar on the complete NVDLA convolution accumulator (CACC), a 25-file hierarchical subsystem with 13 core modules, RAM macros, and utility cells. Cedar decomposes the CACC hierarchically and applies optimization to the three datapath-intensive leaf calculators (INT8, INT16, FP48, each instantiated 64×), then reintegrates the best variants and evaluates whole-design PPA. At 0.9 GHz, the refreshed whole-design flow reduces area by 4.54% (1.044 mm^2 → 0.997 mm^2) and PrimeTime-SAIF power by 0.92%, with unchanged critical-path delay. The FP48 and INT16 leaf rewrites reduce leaf area by 12.17% and 2.61%, respectively; INT8 remains at baseline.

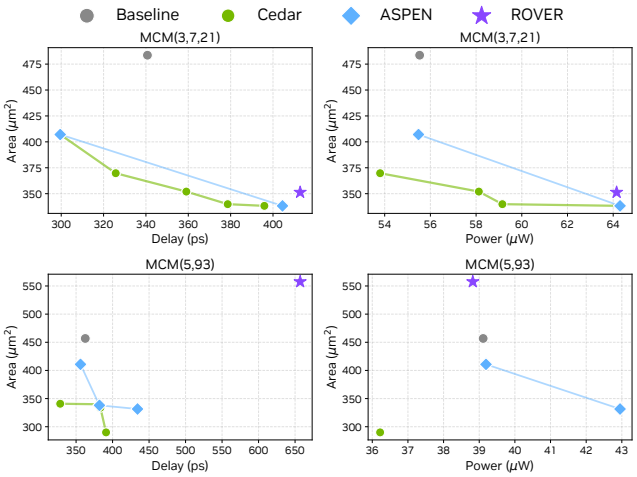


Figure 3: Pareto frontiers on two MCM designs. Cedar achieves similar performance in area/delay trade-off as ASPEN, but improves power.

5.3 Ablation Study

Table 3 isolates the workflow components across 16 designs: eight arithmetic-restructuring designs, four frequency dividers, three control/FSM designs, and one dead-logic mux (13 from RTLLM and three from RTL-OPT). Full Cedar gives the most balanced gains, with the best mean delay and power while nearly matching the direct-rewrite configuration in area. No mechanism dominates every design: direct rewrites benefit arithmetic restructuring, whereas verified-rule e-graphs benefit mux/control-heavy cases. On the Snitch IPU, rules/e-graph alone achieves the full 20.4% area and 45.1% power reduction; the multiplier's 47.1% area gain instead requires the direct rewrite path.

5.4 Comparison with LLM-Based Methods

We compare Cedar against two LLM-based RTL optimization approaches on the RTL-OPT suite: Opus 4.6 direct editing, which applies the published prompt for direct LLM editing, and POET [16], which uses evolutionary LLM search. All three methods are our replications using the same synthesis tool, ASAP7 PDK, and shmoo frequencies, ensuring a consistent comparison. Table 4 presents representative results from the suite.

Arithmetic Designs. Adder and multiplier circuits benefit most: `add_sub` achieves 31.3% area and 24.6% power reduction, while `adder_select` achieves 22.8% area and 38.9% delay reduction through shift-add decomposition and Boolean simplification; `addr_calcu` achieves 55.7% area and 11.9% delay reduction via constant folding and carry-chain simplification. Rules discovered on simple benchmarks apply directly to larger circuits, illustrating the compounding effect of the rule pool.

Control Logic and Multiplexers. FSMs benefit from encoding transformations (`fsm`: 28.6% area, 45.5% power; `fsm_encode`: 19.8% area); mux designs see large gains via cascaded-to-tree restructuring (`mux_dead`: 53.2% area); comparators achieve up to 38.8% delay reduction. These categories exercise the new operator classes beyond prior arithmetic-only support.

Table 4: Representative results on the 40-design RTL-OPT suite (ASAP7 7 nm, shmoo frequency per design). We compare Cedar against Opus 4.6 (RTL-OPT prompt) and POET. Area in μm^2, delay in ps, power in μW (PrimeTime SAIF). ΔA/ΔD/ΔP: best percentage change vs. baseline (negative = improvement, positive = degradation).

		CAD Baseline			Opus 4.6 (RTL-OPT prompt)			POET			Cedar		
Cat.	**Design**	**A (μm^2)**	**D (ps)**	**P (μW)**	ΔA (%)	ΔD (%)	ΔP (%)	ΔA (%)	ΔD (%)	ΔP (%)	ΔA (%)	ΔD (%)	ΔP (%)
Add.	add_sub	282.0	485	24.3	−43.8%	+19.5%	−38.6%	−43.8%	−29.9%	−24.1%	−31.3%	−0.2%	−24.6%
	adder_select	367.9	887	25.7	−22.8%	+3.9%	−23.5%	−22.8%	0.0%	−12.4%	−22.8%	−38.9%	−5.0%
ALU	alu_8bit	184.8	257	18.0	0.0%	0.0%	0.0%	−30.1%	−4.0%	−10.6%	−34.1%	−2.5%	0.0%
	calculation	1137.2	1606	52.1	−21.0%	0.0%	−15.0%	−30.5%	0.0%	−15.9%	−23.5%	0.0%	−15.8%
Cmp.	cmp_8bit	44.3	128	9.1	−47.4%	+27.1%	−48.9%	−47.4%	−15.8%	−19.3%	−42.6%	−20.0%	−14.9%
	cmp_16bit	84.0	191	14.7	−33.9%	+31.6%	−39.7%	−40.6%	−31.7%	−18.9%	−40.6%	−38.8%	−18.9%
FSM	fsm	167.0	187	17.3	−26.8%	−11.4%	−45.2%	−30.0%	−6.2%	−48.0%	−28.6%	−6.8%	−45.5%
Mult.	mac	801.6	626	123.5	0.0%	0.0%	0.0%	0.0%	−12.1%	−17.7%	−31.3%	0.0%	−46.9%
Mux	mux_dead	39.9	46	11.8	−53.2%	−27.7%	−51.4%	−53.2%	−27.7%	−1.3%	−53.2%	−27.7%	−1.3%
	selector	66.5	206	9.0	−18.6%	−34.9%	−16.2%	−18.6%	−31.4%	0.0%	−23.9%	−45.3%	−1.7%
Sub.	sub_4bit	48.3	181	5.2	0.0%	0.0%	0.0%	−7.6%	−31.3%	−2.2%	−2.4%	−17.5%	0.0%
Geo. Mean (40)		—	—	—	−11.9%	−2.9%	−13.2%	−27.3%	−25.1%	−27.9%	−32.3%	−17.1%	−22.8%

POET improves 40/40 designs on at least one metric; strongest on dividers (−57% area) and comparators (−54% area). Cedar achieves 32.3% vs. 27.3% (POET) vs. 11.9% (RTL-OPT) area reduction.

Table 5: Comparison with prior e-graph RTL optimizers on the 7 MLCAD datapath benchmarks (ASAP7 7 nm, shmoo frequency per design). Δ: best % change vs. baseline (negative = improvement), from Pareto frontier (negative = improvement, positive = degradation).

	CAD Baseline			ROVER			ASPEN			Cedar		
Design	**A (μm^2)**	**D (ps)**	**P (μW)**	ΔA (%)	ΔD (%)	ΔP (%)	ΔA (%)	ΔD (%)	ΔP (%)	ΔA (%)	ΔD (%)	ΔP (%)
fir	3213.9	542.7	256.1	−13.3%	−13.6%	0.0%	−26.4%	−7.6%	−2.2%	−26.4%	−8.0%	0.0%
dct	54104.9	1294.1	1367.0	−8.8%	0.0%	0.0%	−56.1%	0.0%	0.0%	−56.5%	−5.1%	−50.2%
polynomial	10204.8	2500.9	409.0	−5.3%	0.0%	−60.8%	−16.0%	0.0%	−0.1%	−19.5%	−0.4%	−60.8%
watermark	166.8	199.5	25.9	0.0%	0.0%	0.0%	−2.7%	−3.9%	0.0%	−3.9%	−52.8%	−8.4%
mcm(3,7,21)	483.6	340.7	55.5	−27.4%	0.0%	0.0%	−30.1%	−12.1%	−0.1%	−30.1%	−12.1%	−3.1%
mcm(7,19,31)	676.0	400.3	46.4	−24.8%	0.0%	0.0%	−39.5%	−15.0%	−4.7%	−41.2%	−15.0%	−4.7%
mcm(5,93)	456.8	362.6	39.1	0.0%	0.0%	−0.7%	−27.4%	−1.8%	0.0%	−36.5%	−9.5%	−7.4%
Geo. Mean	—	—	—	−12.0%	−2.1%	−12.6%	−30.2%	−5.9%	−1.0%	−32.4%	−16.7%	−23.5%

All results are our replications with consistent PrimeTime SAIF power testbench. ROVER [7]: 14 algebraic rules. ASPEN [31]: per-design LLM rules, area+delay only.

Comparison with Baselines. Opus 4.6 achieves 11.9% geometric-mean area reduction but introduces regressions on 3 designs (worst: sub_8bit +14.1% area). POET [16] achieves 27.3% area, 25.1% delay, and 27.9% power reduction, competitive with Cedar on power, but relies on differential testing rather than formal equivalence. Across all 40 designs, Cedar achieves 32.3% area, 17.1% delay, and 22.8% power geometric-mean reduction with no regressions and correctness checks via Z3 rule verification and shipped functional-equivalence tests.

5.5 Comparison with Prior E-Graph Optimizers

Table 5 compares Cedar with ROVER and ASPEN on the 7 MLCAD datapath benchmarks (ASAP7, shmoo frequencies). Cedar achieves −32.4% area (vs. ASPEN −30.2%, ROVER −12.0%), −16.7% delay (vs. −5.9%, −2.1%), and −23.5% power (vs. −1.0%, −12.6%) in geometric mean. The area gap is modest, but Cedar leads on delay and especially power. Power is the key differentiator: ASPEN optimizes only area and delay, so it achieves negligible power improvement (−1.0%), whereas Cedar's joint-PPA cost tuning with SAIF-based power feedback closes this gap. Figure 3 visualizes Pareto frontiers for two designs.

5.6 Cross-Design Rule Transfer

A key claim of Cedar is that rules accumulated on one set of designs transfer to unseen designs. We compare three settings on 9 RTL-OPT designs not used to build the pool; Table 6 shows four representative designs.

From Scratch: starts from 14 initial rules and proposes new rules per design. **Transfer:** uses the frozen 207-rule evaluation snapshot with no per-design rule proposal; the original 55-rule snapshot was unavailable. **Full Pool:** uses the same 207-rule snapshot with per-design proposal enabled (upper bound). All power is measured with PrimeTime SAIF (Section 4.3).

Table 6 shows substantial transferred gains: Transfer achieves −29.1% area versus −31.7% From Scratch, with Full Pool at −30.9%. These are geometric means recomputed from the nine shipped result cells. Rules accumulated on RTLLM designs transfer to unseen RTL-OPT designs without per-design rediscovery, validating the accumulate-and-reuse mechanism.

Table 6: Cross-design rule transfer on 9 unseen RTL-OPT designs; 4 representative designs shown, Mean computed over all 9 (negative = improvement, positive = degradation).

Design	Configuration	ΔArea	ΔDelay	ΔPower
fsm	From Scratch	−33.5%	−7.0%	−49.3%
	Transfer	−30.0%	−6.2%	−45.6%
	Full Pool	−45.8%	−4.9%	−61.3%
calculation	From Scratch	−29.9%	0.0%	−15.8%
	Transfer	−29.9%	0.0%	−20.3%
	Full Pool	−29.9%	0.0%	−15.9%
mux_dead	From Scratch	−53.2%	−27.7%	−5.8%
	Transfer	−53.2%	−27.7%	−1.3%
	Full Pool	−53.2%	−27.7%	0.0%
comp_8bit	From Scratch	−40.1%	−32.1%	−14.4%
	Transfer	−48.4%	−20.0%	−15.4%
	Full Pool	−47.4%	−20.0%	−15.4%
Mean	From Scratch	−31.7%	−18.8%	−25.8%
	Transfer	−29.1%	−16.9%	−15.5%
	Full Pool	−30.9%	−16.7%	−18.1%

From Scratch: 14 initial rules + per-design proposal. Transfer: frozen 207-rule evaluation snapshot, no proposal; the original 55-rule snapshot is unavailable. Full Pool: the same snapshot + proposal. Geometric mean over 9 unseen RTL-OPT designs.

Robustness across implementation settings. Freezing the pool and varying one axis at a time, area-gain retention is 0.62 from ASAP7 7 nm to a 45 nm library (0.79 excluding two saturated designs) and 1.00 across TT/FF/SS corners and RVT/LVT cells. At clock constraints $0.8D$ and $1.2D$, area retention is 1.04/0.83 and power retention is 0.78/0.63; flattening retains 0.92 and enabling datapath optimization retains 0.58. Transfer to Yosys+ABC is tool-sensitive, retaining 0.30 of the commercial-tool area gain.

6 Discussion

6.1 Rule Pool Dynamics

The pattern-match, historical-delta, and conflict-avoidance criteria (Section 3.2) keep per-design rule counts tractable (8 to 15 from the 207-rule evaluation snapshot) while maximizing coverage. The pool's growth is not redundant accumulation: as Section 6.2 shows, 9 of 10 sampled rules fill genuine gaps in commercial synthesis, either through structural fidelity or genuinely novel transformations. Growth rate decelerates as common idioms are covered: the pool gains roughly 25 rules per 10 new RTLLM designs early and drops to roughly 5 per 10 by the end of the 44-design pass, indicating near-saturation of recurring patterns. Adding power to the cost model reveals trade-offs invisible to area-only optimization: the Snitch IPU achieves 23.9% post-PnR area and 46.1% power reduction jointly, demonstrating the value of PPA-aware extraction.

6.2 Why Do Local Rewrites Help?

To test whether commercial synthesis already applies our rewrites, we selected 10 representative rules (spanning all four categories) and synthesized minimal Verilog module pairs (isolating each rewrite pattern) with the commercial tool on ASAP7, then compared with impact on real designs. Among these 10 rules, three outcomes emerge. **Structural fidelity (5/10):** The tool applies the rewrite on isolated modules but preserves the surrounding RTL structure in larger designs, yielding 10 to 80% area reduction (e.g., −80% on `comparator_8bit` by restructuring chained comparisons into XOR trees). **Genuinely novel (4/10):** Constant-multiplication decomposition, mux-arithmetic interchange, add/sub unification, and multiplication factoring are outside the tool's repertoire even on trivially small modules. **Context-dependent (1/10):** `sub_to_add_neg` is +60% area in isolation but enables −45% area via adder sharing, motivating equality saturation over greedy rewriting (Figure 1). This characterization is qualitative over these 10 rules and should not be read as a statistical claim over the full 175-rule pool.

Table 7: Instrumented runtime and LLM usage per run.

Design	Time	Calls	Tokens (M)	Cost ($)
accu	42 min	29	0.21	13.09
adder_pipe	1.75 h	41	0.48	24.79
Snitch IPU	1.70 h	141	0.45	36.09
ticket_machine	1.78 h	133	0.49	32.47

Runtime, reproducibility, and verification. Table 7 reports measured per-run cost for four representative designs. Across at least three independent runs per design, every reported best result lies within two standard deviations. Of 215 proposed rules, Z3 accepted 180 (84%); 5 were invalid, 29 unknown, and 1 malformed. With a 60-second budget per rule, 207 finish within one second; rejection is bounded and cannot stall the workflow.

6.3 Limitations

Two of 44 RTLLM designs see no improvement because their sequential structures expose no remaining combinational optimization surface. Cedar treats registers as optimization boundaries; cross-register transformations would require retiming, which we do not attempt. Saturation also has a scalability ceiling: the Snitch IPU's 1,612-node e-graph is near practical limits. Finally, the learned rule pool is calibrated to ASAP7 and may need re-calibration for other technology nodes.

7 Conclusion

Cedar demonstrates that equality saturation scales to full RTL designs an order of magnitude beyond prior work when an LLM agent proposes verified rewrite rules and accumulates them into a growing pool. Rules transfer across design families without per-design rediscovery, and the flow jointly optimizes power, delay, and area while preserving sequential structure under the shipped verification and functional-equivalence checks. Future directions include hierarchical decomposition for million-gate designs and Bayesian optimization of PPA extraction weights.

Acknowledgments

This work was supported in part by ACE, one of the seven centers in JUMP 2.0, a Semiconductor Research Corporation (SRC) program sponsored by DARPA, NSF Award #2403135, #2349670, and #2403134 and a research gift from NVIDIA.

References

[1] Che-Ming Chang, Prashanth Vijayaraghavan, Ashutosh Jadhav, Charles Mackin, Vandana Mukherjee, Hsinyu Tsai, and Ehsan Degan. 2026. CODMAS: A Dialectic Multi-Agent Collaborative Framework for Structured RTL Optimization. arXiv:2603.17204 [cs.AR]

[2] Chao Chen, Guangyu Hu, Cunxi Yu, Yuzhe Ma, and Hongce Zhang. 2025. E-morphic: Scalable Equality Saturation for Structural Exploration in Logic Synthesis. In *Proceedings of the 62nd ACM/IEEE Design Automation Conference (DAC)*. IEEE, San Francisco, CA, USA, 6 pages.

[3] Chao Chen, Guangyu Hu, Duo Zuo, Cunxi Yu, Yuzhe Ma, and Hongce Zhang. 2024. E-syn: E-Graph Rewriting with Technology-Aware Cost Functions for Logic Synthesis. In *Proceedings of the 61st ACM/IEEE Design Automation Conference (DAC)*. IEEE, San Francisco, CA, USA, 1–6.

[4] Jianyi Cheng, Samuel Coward, Lorenzo Chelini, Rafael Barbalho, and Thomas Drane. 2024. Seer: Super-Optimization Explorer for High-Level Synthesis Using E-Graph Rewriting. In *Proceedings of the 29th ACM International Conference on Architectural Support for Programming Languages and Operating Systems*, Vol. 2. ACM, La Jolla, CA, USA, 1029–1044.

[5] Lawrence T. Clark, Vinay Vashishtha, Lucian Shifren, Aditya Gujja, Saurabh Sinha, Brian Cline, Chandarasekaran Ramamurthy, and Greg Yeric. 2016. ASAP7: A 7-nm FinFET Predictive Process Design Kit. *Microelectronics Journal* 53 (2016), 105–115.

[6] Samuel Coward, George A. Constantinides, and Thomas Drane. 2023. Automating Constraint-Aware Datapath Optimization Using E-Graphs. In *Proceedings of the 60th ACM/IEEE Design Automation Conference (DAC)*. ACM/IEEE, San Francisco, CA, USA, 1–6.

[7] Samuel Coward, Thomas Drane, and George A. Constantinides. 2024. ROVER: RTL Optimization via Verified E-Graph Rewriting. *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems* 43, 12 (2024), 4687–4700.

[8] Samuel Coward, Theo Drane, and George A. Constantinides. 2025. Constraint-Aware E-Graph Rewriting for Hardware Performance Optimization. *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems* 44, 4 (2025), 1366–1379.

[9] Samuel Coward, Theo Drane, Emiliano Morini, and George A. Constantinides. 2024. Combining Power and Arithmetic Optimization via Datapath Rewriting. In *2024 IEEE 31st Symposium on Computer Arithmetic (ARITH)*. IEEE, Malaga, Spain, 24–31.

[10] Mingjie Liu, Nathaniel Pinckney, Brucek Khailany, and Haoxing Ren. 2023. VerilogEval: Evaluating Large Language Models for Verilog Code Generation. In *Proceedings of the IEEE/ACM International Conference on Computer-Aided Design (ICCAD)*. IEEE, San Francisco, CA, USA, 1–8.

[11] Shang Liu, Weiguo Fang, Yao Lu, Qijun Zhang, Hongce Zhang, and Zhiyao Xie. 2024. RTLCoder: Outperforming GPT-3.5 in Design RTL Generation with Our Open-Source Dataset and Lightweight Solution. In *2024 IEEE LLM Aided Design Workshop (LAD)*. IEEE, San Jose, CA, USA, 1–5.

[12] Shang Liu, Yao Lu, Wenji Fang, Mengming Li, and Zhiyao Xie. 2024. OpenLLM-RTL: Open Dataset and Benchmark for LLM-Aided Design RTL Generation (Invited). In *IEEE/ACM International Conference on Computer Aided Design (ICCAD)*. IEEE/ACM, New York, NY, USA, 1–9.

[13] LLVM Project. 2024. CIRCT: Circuit IR Compilers and Tools. https://circt.llvm.org/. Part of the LLVM project.

[14] Yao Lu, Shang Liu, Qijun Zhang, Hongce Zhang, and Zhiyao Xie. 2024. RTLLM: An Open-Source Benchmark for Design RTL Generation with Large Language Model. In *29th Asia and South Pacific Design Automation Conference (ASP-DAC)*. IEEE, Incheon, Republic of Korea, 722–727.

[15] NVIDIA. 2024. NVDLA: NVIDIA Deep Learning Accelerator. https://github.com/nvdla/hw. Open-source hardware design.

[16] 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:2603.19333 [cs.AR]

[17] Samuel Riedel, Matheus Cavalcante, Renzo Andri, and Luca Benini. 2023. MemPool: A Scalable Manycore Architecture With a Low-Latency Shared L1 Memory. *IEEE Trans. Comput.* 72, 12 (2023), 3561–3575. doi:10.1109/TC.2023.3307796

[18] Kimia Tasnia, Alexander Garcia, Tasnuva Farheen, and Sazadur Rahman. 2025. VeriOpt: PPA-Aware High-Quality Verilog Generation via Multi-Role LLMs. arXiv:2507.14776 [cs.AR]

[19] Shailja Thakur, Baleegh Ahmad, Hammond Pearce, Benjamin Tan, Brendan Dolan-Gavitt, Ramesh Karri, and Siddharth Garg. 2024. Verigen: A Large Language Model for Verilog Code Generation. *ACM Transactions on Design Automation of Electronic Systems* 29, 3 (2024), 1–31.

[20] Shailja Thakur, Jason Blocklove, Hammond Pearce, Benjamin Tan, Siddharth Garg, and Ramesh Karri. 2023. AutoChip: Automating HDL Generation Using LLM Feedback. arXiv:2311.04887 [cs.PL]

[21] Yun-Da Tsai, Mingjie Liu, and Haoxing Ren. 2024. RTLFixer: Automatically Fixing RTL Syntax Errors with Large Language Models. In *Proceedings of the 61st ACM/IEEE Design Automation Conference*. ACM/IEEE, San Francisco, CA, USA, 1–6.

[22] Mubashir ul Islam, Humza Sami, Pierre-Emmanuel Gaillardon, and Valerio Tenace. 2025. EDA-Aware RTL Generation with Large Language Models. In *2025 Design, Automation & Test in Europe Conference (DATE)*. IEEE, Lyon, France, 1–6.

[23] Ecenur Ustun, Cunxi Yu, and Zhiru Zhang. 2023. Equality Saturation for Datapath Synthesis: A Pathway to Pareto Optimality. In *Proceedings of the 60th ACM/IEEE Design Automation Conference (DAC)*. IEEE, San Francisco, CA, USA, 1–2.

[24] Alexa VanHattum, Rachit Nigam, Vincent T. Lee, James Bornholt, and Adrian Sampson. 2021. Vectorization for Digital Signal Processors via Equality Saturation. In *Proceedings of the 26th ACM International Conference on Architectural Support for Programming Languages and Operating Systems*. ACM, Virtual, USA, 874–886.

[25] 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. arXiv:2504.10369 [cs.AR] NeurIPS 2025 poster.

[26] Yisu Remy Wang, Shana Hutchison, Jonathan Leang, Bill Howe, and Dan Suciu. 2020. SPORES: Sum-Product Optimization via Relational Equality Saturation for Large-Scale Linear Algebra. arXiv:2002.07951 [cs.DB]

[27] Anjiang Wei, Huanmi Tan, Tarun Suresh, Daniel Mendoza, Thiago SFX Teixeira, Ke Wang, Caroline Trippel, and Alex Aiken. 2025. VeriCoder: Enhancing LLM-Based RTL Code Generation through Functional Correctness Validation. NeurIPS 2025 Fourth Workshop on Deep Learning for Code.

[28] Max Willsey, Chandrakana Nandi, Yisu Remy Wang, Oliver Flatt, Zachary Tatlock, and Pavel Panchekha. 2021. egg: Fast and Extensible Equality Saturation. *Proceedings of the ACM on Programming Languages* 5, POPL (2021), 1–29.

[29] Jiaqi Yin, Zhufei Song, Qirun Chen, Guangyu Hu, and Cunxi Yu. 2025. BoolE: Exact Symbolic Reasoning via Boolean Equality Saturation. In *Proceedings of the 62nd ACM/IEEE Design Automation Conference (DAC)*. IEEE, San Francisco, CA, USA, 6 pages.

[30] Florian Zaruba, Fabian Schuiki, Torsten Hoefler, and Luca Benini. 2021. Snitch: A Tiny Pseudo Dual-Issue Processor for Area and Energy Efficient Execution of Floating-Point Intensive Workloads. *IEEE Trans. Comput.* 70, 11 (2021), 1845–1860.

[31] Niansong Zhang, Chenhui Deng, Johannes Maximilian Kuehn, Chia-Tung Ho, Cunxi Yu, Zhiru Zhang, and Haoxing Ren. 2025. ASPEN: LLM-Guided E-Graph Rewriting for RTL Datapath Optimization. In *Proceedings of the ACM/IEEE Workshop on Machine Learning for CAD (MLCAD)*. IEEE, Santa Cruz, CA, USA, 1–8.

[32] Yihong R. Zhang, Yisu Remy Wang, Oliver Flatt, David Cao, Philip Zucker, Eli Rosenthal, Zachary Tatlock, and Max Willsey. 2023. Better Together: Unifying Datalog and Equality Saturation. *Proceedings of the ACM on Programming Languages* 7, PLDI (2023), 468–492.

[33] Hao Zhao, Hanqi Zhang, Zhi Huang, Jiaxi Yu, and Jieru Zhao. 2024. MAGE: A Multi-Agent Engine for Automated RTL Code Generation. arXiv:2412.07822 [cs.AR]