# TURBORTL: RIGOROUS EVALUATION OF LLMS AND AGENTS FOR RTL OPTIMIZATION

**Anonymous authors**
Paper under double-blind review

## ABSTRACT

Although large language models (LLMs) have demonstrated growing capabilities in Verilog code generation, existing research largely prioritizes syntactical and functional correctness, overlooking the critical hardware design goal of optimizing circuit efficiency. Consequently, the field lacks a well-established benchmark to rigorously assess the ability of LLMs to optimize register-transfer level (RTL) designs. To this end, we introduce **TurboRTL**, a comprehensive Verilog code optimization benchmark constructed via an automated pipeline that utilizes equivalence graph (e-graph) rewriting to systematically identify designs with significant optimization potential. Furthermore, reflecting the reality that hardware engineers optimize designs through iterative refinement rather than single-shot attempts, our evaluation framework extends beyond individual models to incorporate two tool-augmented interaction paradigms, specifically population-based test-time search and experience-driven agentic workflows. We demonstrate that while individual models show initial promise, these advanced agentic strategies are essential for tackling the complexities of this domain. By providing both a high-quality benchmark and a diverse evaluation pipeline, this work establishes a critical foundation for advancing automated RTL optimization.

## 1 INTRODUCTION

The integration of large language models (LLMs) into electronic design automation (EDA) has reshaped hardware design workflows (Chen et al., 2024). Recent years have seen extensive work automating register-transfer level (RTL) code generation (Liu et al., 2024; Pei et al., 2024; Zhao et al., 2025), supported by benchmarks such as VerilogEval (Liu et al., 2023b) and RTLLM (Lu et al., 2024). However, these approaches prioritize syntactic and functional correctness over the quality of resulting designs, which is central to hardware engineering. The bottleneck is RTL *optimization*: in standard workflows, the initial implementation is only a baseline, while most effort goes into iterative refinement under stringent efficiency constraints. Yet this important phase remains beyond current LLM methods because rigorous, representative benchmarks are lacking.

RTL optimization exposes two fundamental challenges that reveal the limitations of current LLMs. First, next-token training captures local syntax, whereas optimization requires semantic understanding of a design's underlying structure to refactor logic or enable resource sharing without violating functional equivalence. Second, performance metrics (*e.g.*, area and delay) are measured on the synthesized netlist, not the RTL source. Unlike software, where source efficiency often correlates with execution speed, hardware quality is determined only after mapping RTL to a technology library through logic synthesis. Models must therefore generate designs with superior synthesized implementations, rather than merely producing textually concise code.

Recent RTL optimization benchmarks remain limited in validity and scale. A meaningful benchmark must establish intrinsic *optimization potential*: a superior implementation must exist beyond standard synthesis. RTLRewriter (Yao et al., 2024) creates *suboptimal* baselines by inserting redundant arithmetic (*e.g.*, adding zero or multiplying by one), but commercial tools such as Synopsys Design Compiler can trivially remove these artifacts via `compile_ultra`. Such benchmarks therefore test denoising of artificial artifacts more than meaningful optimization that survives synthesis. Pluto (Abdelatty et al., 2025) and RTL-OPT (Lu et al., 2026) use human experts to craft optimized designs, but



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

manual curation limits the resulting benchmark's scale and scope. They also often draw on simpler modules from VerilogEval and RTLLM, reducing representativeness of complex designs.

To bridge this gap, we introduce **TurboRTL**, a comprehensive Verilog optimization benchmark built through an automated pipeline using equivalence-graph (e-graph) rewriting. E-graphs compactly represent many functionally equivalent programs through non-destructive rewrite rules, a technique traditionally used in compiler optimization (Tate et al., 2009; Willsey et al., 2021). We repurpose this methodology to strictly validate benchmark candidates. Unlike prior work constrained by manual curation, applying equality saturation to thousands of open-source RTL designs yields a substantially more diverse and complex design set. We retain only designs for which the e-graph solver finds a mathematically equivalent variant with a lower area-delay product (ADP), even under the strict optimization constraints of the `compile_ultra` flow. Thus, every TurboRTL design has proven, non-trivial optimization potential beyond commercial synthesis, establishing a rigorous and reproducible optimization target for model evaluation.

Furthermore, prior evaluations typically rely on single-shot prompting and miss the iterative nature of hardware design. Hardware engineers instead refine designs iteratively using EDA feedback rather than through isolated attempts. Our framework therefore evaluates two tool-augmented paradigms: population-based test-time search, which uses evolutionary mechanisms to explore structural innovations, and experience-driven agentic workflows, which iteratively refine strategies through experience management. These settings more closely reflect the feedback-driven optimization process used in practical hardware development. We show that although individual models show initial promise, these advanced agentic strategies are essential for tackling the domain's complexity. Together, the high-quality benchmark and diverse evaluation pipeline establish a foundation for advancing the frontier of automated RTL optimization.

## 2 Related work

### 2.1 LLM-based RTL generation

RTL is a critical abstraction in EDA that describes the flow of data between registers and the logical operations performed on that data. This level is typically expressed using hardware description languages (HDLs), with Verilog being the most widely used in the industry. Consequently, the terms RTL and Verilog are used interchangeably in this work. The application of LLMs to hardware design has primarily focused on generating RTL code from natural language specifications. Foundational benchmarks such as VerilogEval (Liu et al., 2023b) and RTLLM (Lu et al., 2024) establish the standard for this domain, utilizing metrics like `pass@k` (Chen et al., 2021) to measure syntactic and functional correctness. To address the limitations of general-purpose models, numerous recent studies (Liu et al., 2023a; Thakur et al., 2024; Liu et al., 2024; Pei et al., 2024; Liu et al., 2025b;a;c; Xu et al., 2025; Zhao et al., 2025) leverage domain-specific fine-tuning and multi-objective training to significantly improve generation quality. Subsequently, advanced agentic frameworks (Ho et al., 2025) have incorporated tool-feedback loops, pushing functional correctness rates on established benchmarks to over 94%. However, despite these advances, existing approaches largely prioritize syntactical and functional correctness while overlooking the quality of the resulting designs. In standard hardware design workflows, the initial implementation serves merely as a baseline; the majority of effort is dedicated to iterative refinement to meet stringent efficiency constraints. Consequently, the critical task of *optimizing* RTL designs remains a significant bottleneck for current LLM methodologies.

### 2.2 Existing benchmarks for RTL optimization

Recently, several efforts to benchmark RTL optimization have emerged, yet existing datasets suffer from critical limitations in validity and scale. RTLRewriter (Yao et al., 2024) introduces a retrieval-augmented optimization framework, but its dataset relies on *suboptimal* baselines constructed via artificial noise injection (*e.g.*, adding zero or redundant wire assignments). These contrived inefficiencies are trivially eliminated by commercial synthesis tools (*e.g.*, Synopsys Design Compiler), rendering the benchmark a test of text denoising rather than actual optimization. Subsequent benchmarks like RTL-OPT (Lu et al., 2026) (36 designs) and Pluto (Abdelatty et al., 2025) (114 designs) address this validity issue by employing human experts to handcraft optimized designs. However, reliance on manual curation fundamentally limits scalability, resulting in small datasets often adapted

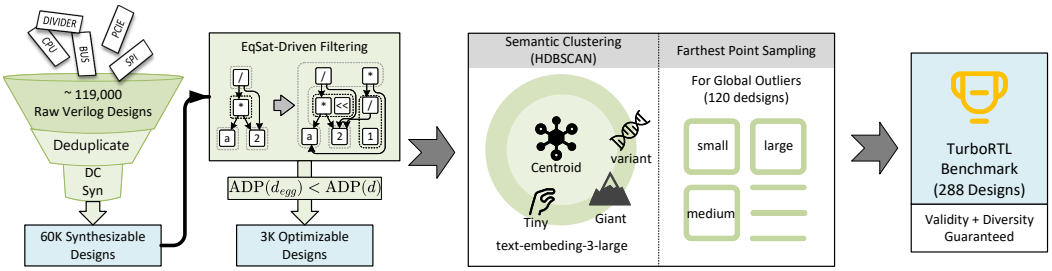



Figure 1: Overview of the TurboRTL construction pipeline. The workflow proceeds in three stages: (1) EqSat-Driven Filtering, which filters ∼119k raw files down to designs with proven non-trivial optimization potential via e-graph rewriting and aggressive synthesis checks; (2) Semantic Clustering, which organizes validated designs into functional groups using HDBSCAN and LLM-based consistency checks; and (3) Stratified Extremes Sampling, which ensures diversity by selecting four distinct implementation styles per cluster (Archetype, Tiny, Giant, Variant) alongside global outliers, yielding the final 288-design benchmark.

from previous simple code generation benchmarks (*e.g.*, VerilogEval, RTLLM), thereby restricting their representativeness of diverse and complex designs. TurboRTL distinguishes itself through an automated pipeline: we mine thousands of designs and use formal methods to filter them, ensuring a benchmark that is both large-scale and demonstrates non-trivial optimization potential. Moreover, we significantly expand the evaluation scope. While prior works largely restrict evaluation to single-shot prompting, with RTL-OPT notably limiting evaluation to a single query per model, TurboRTL explicitly benchmarks population-based test-time search and experience-driven agentic workflows, reflecting the iterative nature of the hardware design process.

### 2.3 E-GRAPH REWRITING FOR PROGRAM OPTIMIZATION

E-graph rewriting has emerged as a transformative technique for program optimization, finding diverse applications such as optimizing compiler intermediate representations and automating deep learning graph fusion (Willsey et al., 2021). At its core, an e-graph compactly represents an exponential space of functionally equivalent programs by grouping semantically identical expressions into equivalence classes. This structure enables equality saturation (EqSat) (Tate et al., 2009), a process that iteratively applies rewrite rules to explore the design space non-destructively until saturation, thereby avoiding the local optima common in traditional destructive compilers. While prior works have utilized EqSat as an *optimizer* to directly improve datapath circuits (Coward et al., 2024), we repurpose it as a validation mechanism. By saturating the e-graph for thousands of mined designs, we strictly select only those where the e-graph solver proves a lower-cost variant exists, even under rigorous commercial synthesis settings. This guarantees that every design in TurboRTL possesses proven non-trivial optimization potential, while the automated nature of the pipeline enables the inclusion of significantly more diverse and complex designs.

## 3 BENCHMARK CONSTRUCTION

We define the RTL optimization task as utilizing an LLM to transform an input design $d$ into a functionally equivalent variant $d'$ such that $ADP(d') < ADP(d)$. To rigorously benchmark this capability, each design must be proven to possess intrinsic optimization potential, *i.e.*, a more efficient implementation is known to exist. To strictly guarantee this validity alongside diversity (coverage of functional types and coding styles), we construct TurboRTL via a multi-stage automated pipeline. This process filters a corpus of over 100,000 raw designs down to a curated set of 288 high-quality designs, each validated to demonstrate non-trivial optimization potential.

### 3.1 E-GRAPH REWRITING FOR DATA FILTERING

**Data collection and pre-processing.** We assemble data from existing hardware design datasets, including RTLCoder (Liu et al., 2024), MG-Verilog (Zhang et al., 2024), DeepRTL (Liu et al., 2025b),

and DeepCircuitX (Li et al., 2025), resulting in a raw corpus of 118,954 Verilog files, each typically consisting of a single module. To ensure data quality, we deduplicate the corpus using MinHash and Jaccard similarity metrics (Yan et al., 2017). Furthermore, we enforce a strict synthesis pre-check using Synopsys Design Compiler (DC), retaining only those designs that yield meaningful area and delay metrics to eliminate incomplete or non-synthesizable designs. This rigorous filtering process produces a candidate set, $\mathcal{D}_{cand}$, of approximately 60,000 modules.

**The EqSat-driven filter.** To ensure the benchmark samples capture non-trivial optimization potential, we employ a filtering process that integrates e-graph rewriting with industrial-grade synthesis metrics. We first parse each candidate design $d \in \mathcal{D}_{cand}$ into S-expressions, a prefix-form symbolic expression suitable for e-graph rewriting that encodes the design's abstract syntax tree (AST), using the Stagira Verilog parser (Chen et al., 2023). We then utilize the `egg` library (Willsey et al., 2021) to perform EqSat. The rewriting process utilizes a comprehensive set of 76 Verilog-specific rewrite rules curated from the RuleNet framework (Authors, 2027), which are derived from verified algebraic identities, compiler transformations, and practical hardware rewrites. Following EqSat, we extract the optimal variant, $d_{egg}$, by utilizing the ADP as the cost function. We purposefully exclude power consumption from this cost function, as reliable dynamic power analysis requires representative stimuli that are often unavailable for general designs; a detailed justification for this choice is provided in Appendix D. We then synthesize both the original design and each e-graph-derived candidate with Synopsys Design Compiler using the `compile_ultra` command, which applies aggressive logic optimization and technology mapping to estimate the resulting circuit's area and delay under a target technology library. We use Design Compiler Q-2019.12-SP4 with the SkyWater 130 nm `sky130_fd_sc_hd__tt_025C_1v80` library and apply exactly the same flow to every design being compared. The purpose of this step is to ensure that the measured optimization is not merely a superficial RTL simplification: any redundancy that the commercial compiler can remove on its own should disappear in both versions, and should not be counted as meaningful optimization. We therefore retain only improvements that survive this common post-synthesis flow and yield a lower ADP for the e-graph-derived design. Finally, we utilize Yosys (Wolf et al., 2013) to formally verify the functional equivalence between the original design $d$ and the optimized variant $d_{egg}$. A design is retained in the benchmark if and only if the optimal variant demonstrates a strictly superior ADP after aggressive synthesis:

$$ADP(d_{egg}) < ADP(d) \tag{1}$$

This process reduces the dataset to 3,414 designs, ensuring that every benchmark sample possesses a mathematically proven path to enhanced efficiency that surpasses the capabilities of state-of-the-art commercial compilers. Appendix E presents more details regarding the e-graph rewriting process.

### 3.2 Semantic clustering and consistency check

Following EqSat-driven filtering, the candidate set still comprises 3,414 designs. This volume remains computationally prohibitive for extensive benchmarking and is prone to structural redundancy (*e.g.*, over-represented basic counters). To refine this corpus into a compact, high-quality, and functionally diverse benchmark, we employ a semantic clustering approach. We generate vector embeddings $\mathbf{v}_d$ for the validated designs $d$ using OpenAI's `text-embedding-3-large` (Neelakantan et al., 2022) and apply HDBSCAN (McInnes et al., 2017), a hierarchical density-based clustering algorithm. To validate cluster coherence while minimizing computational overhead, we employ a "**Centroid + Boundary**" **Consistency Check**. For each cluster, we select 3 *Core* designs (highest probability density) and 2 *Edge* designs (lowest non-zero probability). We then prompt two advanced models, GPT-5.2 (OpenAI, 2025) and Gemini-3-pro (Google DeepMind, 2025), to generate a functional label and a consistency score (0–10). We take the average of these scores to iteratively fine-tune the HDBSCAN hyperparameters. By setting the minimum cluster size to 8 and minimum samples to 4, we achieve a final average consistency score of 7.2, yielding a total of 62 distinct functional clusters.

### 3.3 Stratified extremes sampling

To curate the final benchmark, we employ a stratified extremes sampling strategy to evaluate LLMs not just on diverse functions but across varying implementation styles and design sizes. As illustrated in Figure 1, we first restrict our selection to functional clusters exhibiting a high consistency score ($\geq 6$), dismissing low-confidence groupings as noise. For each validated cluster $C_k$, we select four distinct representatives. We identify the **Archetype (Centroid)** as the design mathematically

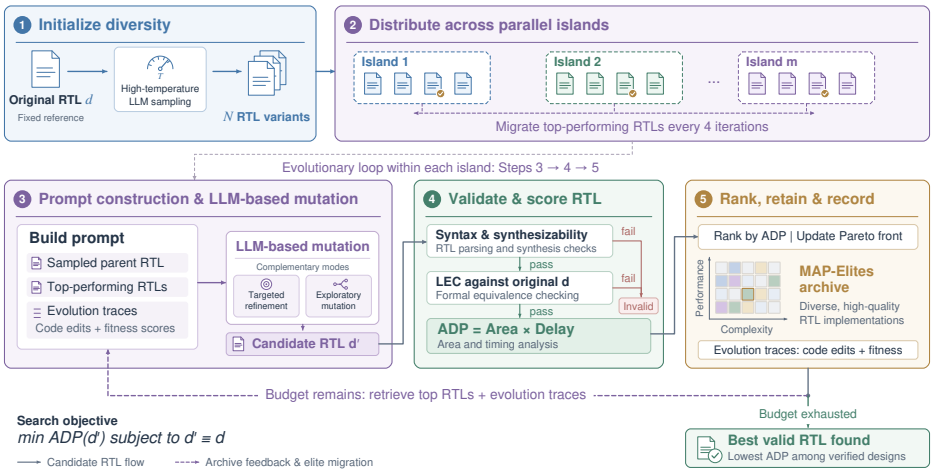



Figure 2: Overview of the population-based test-time search for RTL optimization. High-temperature sampling initializes diverse RTL variants, which evolve across parallel islands through LLM-based mutation. Candidates are checked for synthesizability and equivalence, scored by ADP, and retained in a MAP-Elites archive until the query budget is exhausted.

closest to the cluster center ($\text{argmin}_{d \in C_k} \|\mathbf{v}_d - \mu_k\|$), where $\mu_k = \frac{1}{|C_k|} \sum_{d \in C_k} \mathbf{v}_d$, representing the standard implementation. To assess robustness across design scales, we select the **Tiny** and **Giant** variants—designs with the minimum and maximum token counts, respectively, utilizing the `tiktoken cl100k_base` encoding (OpenAI, 2023) for measurement. Finally, we include the **Variant (Structural Outlier)**, defined as the design with the maximum Euclidean distance from the centroid ($\text{argmax}_{d \in C_k} \|\mathbf{v}_d - \mu_k\|$), capturing the "same function, different structure" paradigm.

**Global outlier sampling.** To capture unique logic that defies standard categorization, we also sample from the noise points identified by HDBSCAN, including designs from dismissed low-confidence clusters. We stratify these points into three bins of uniform population based on token count quantiles and apply farthest point sampling (FPS) (Hochbaum & Shmoys, 1985) to select 120 diverse outliers. Combining these with the 168 cluster representatives, we curate the final TurboRTL benchmark of 288 designs, providing a comprehensive testbed for automated RTL optimization. A comprehensive quantitative comparison between TurboRTL and prior work is provided in Appendix F.

## 4 Evaluation Framework

To comprehensively assess the capabilities of LLMs for RTL optimization, we design a framework that evaluates models using three distinct interaction paradigms. First, we define a rigorous set of progressive metrics for single-shot generation evaluation, aiming to quantify the intrinsic optimization potential of base models. Second, we examine population-based test-time search, evaluating how evolutionary mechanisms can navigate the design space for structural innovations. Finally, we explore experience-driven agentic workflows, which utilize experience management to iteratively refine optimization strategies over time. Crucially, both of these advanced paradigms incorporate iterative feedback from EDA tools, enabling a closed-loop optimization process guided by hardware performance metrics. This structure ensures a holistic evaluation, encompassing intrinsic generation, evolutionary search, and adaptive refinement.

### 4.1 Single-Shot Generation Evaluation

In the domain of RTL optimization, evaluating model performance requires extending standard code generation metrics. Beyond generating syntactically valid code, models must ensure synthesizability and functional equivalence while delivering quantifiable performance gains. To rigorously assess this, we define four progressive metrics (*i.e.*, `syn@k`, `fun@k`, `opt@k`, and `eff@k`) that capture a model's ability to navigate the hardware optimization landscape.

The first three metrics (`syn@k`, `fun@k`, and `opt@k`) measure the frequency of success under increasingly stringent criteria. Following the standard pass@k formulation (Chen et al., 2021), we calculate these metrics using an unbiased estimator for the probability that at least one sample in a generated set of size $k$ satisfies a specific criterion $\Phi_{\text{metric}}$. Given $n$ total generated samples where $c$ samples satisfy the criterion, the metric is calculated as:

$$\text{Metric@}k := \mathbb{E}_{\text{problems}}\left[1 - \frac{\binom{n-c}{k}}{\binom{n}{k}}\right] \tag{2}$$

In our experiments, we set $n = 20$ to mitigate randomness and report results for $k \in \{1, 5, 10\}$. Let $s$ denote a generated code sample and $d$ the original reference design. The predicates for each metric are defined as follows:

- $\Phi_{\text{syn}}(s) := \mathbb{I}(\text{Synthesis}(s) = \text{Success})$, ensuring that the design can be successfully synthesized.
- $\Phi_{\text{fun}}(s) := \Phi_{\text{syn}}(s) \wedge \mathbb{I}(\text{LEC}(s, d) = \text{True})$, ensuring logical equivalence to the reference design.
- $\Phi_{\text{opt}}(s) := \Phi_{\text{fun}}(s) \wedge \mathbb{I}(\text{ADP}(s) < \text{ADP}(d))$, ensuring a strictly lower ADP.

While the above metrics quantify success frequency, they fail to capture the magnitude of optimization. To address this limitation, we adopt the `eff@k` (Efficiency Ratio) metric from the ENAMEL benchmark (Qiu et al., 2025). Unlike binary success metrics, `eff@k` measures the expected maximum normalized improvement achieved within a given sample budget. We first define the efficiency score $e_i$ for the $i$-th sample $s_i$ as the degree to which it closes the gap between the baseline design $d$ and the performance target established by the optimal variant $d_{egg}$ obtained via EqSat-driven filtering:

$$e_i = \begin{cases} \frac{\text{ADP}(d) - \text{ADP}(s_i)}{\text{ADP}(d) - \text{ADP}(d_{egg})} & \text{if } \Phi_{\text{opt}}(s_i) \text{ is True} \\ 0 & \text{otherwise} \end{cases} \tag{3}$$

Under this definition, samples that are non-synthesizable, functionally inequivalent, or fail to improve upon the baseline result in a score of 0. The `eff@k` is calculated as the expected maximum efficiency score over a random subset $S_k$ of size $k$ drawn from the total pool of $n$ generated samples $S_n$:

$$\begin{aligned} \text{eff@}k &= \mathbb{E}_{\text{problems}}\left[\mathbb{E}_{S_k \subset S_n}\left[\max_{s \in S_k} e(s)\right]\right] \\ &= \mathbb{E}_{\text{problems}}\left[\sum_{r=k}^{n} \frac{\binom{r-1}{k-1}}{\binom{n}{k}} e_{(r)}\right] \end{aligned} \tag{4}$$

where $e_{(r)}$ denotes the $r$-th smallest efficiency score among the $n$ samples for a given problem. A score of $1.0$ indicates the LLM matches the optimal structure identified by the e-graph rewriting process, whereas a score of $0.0$ indicates no improvement over the original baseline.

### 4.2 Population-based test-time search

We implement a population-based test-time search to navigate the vast design space for structural innovations, adapting the AlphaEvolve (Novikov et al., 2025) framework to the RTL optimization domain. While single-shot generation relies on a single query to generate an optimized design, our population-based strategy treats optimization as a dynamic, tool-augmented evolutionary search. As illustrated in Figure 2, the process begins by spawning $N$ RTL variants using high-temperature LLM sampling. To ensure scalability and prevent premature convergence, we utilize an island-based architecture where isolated subpopulations evolve in parallel. Within this framework, the LLM acts as a genetic mutation operator, leveraging iterative feedback from EDA tools to drive both targeted refinements and exploratory mutations. Each variant is evaluated and stored in a MAP-Elites archive, a grid-based database that indexes designs by complexity and performance to maintain a diverse set of high-quality solutions. The system iterates for 20 generations, continuously updating the Pareto front of optimized designs. Every four iterations, island migration occurs, allowing top-performing RTL implementations to propagate across subpopulations. By maintaining a diverse population and utilizing tool-guided mutations, this search strategy enables a global exploration of structural transformations that would be impossible to achieve through a single-shot attempt.

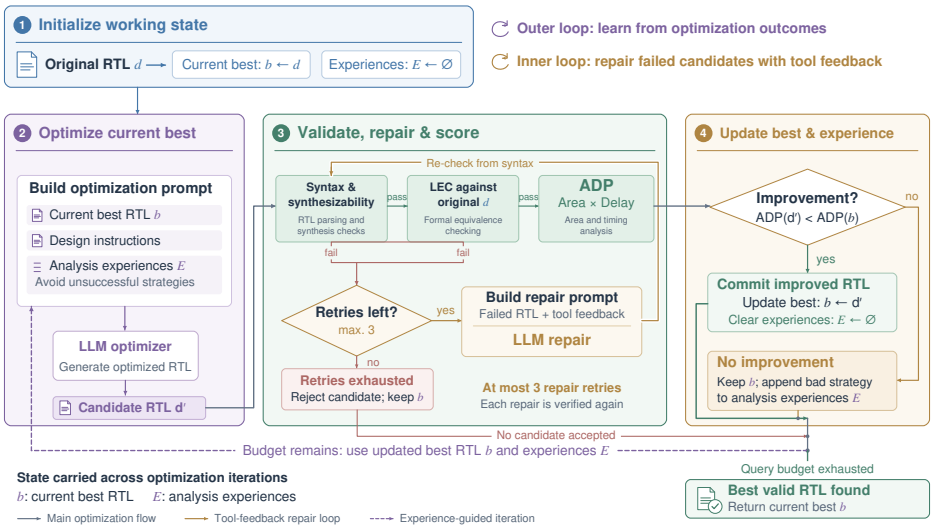



Figure 3: Overview of the experience-driven agentic workflow for RTL optimization. The LLM optimizes the current best RTL using accumulated experiences, while synthesis and equivalence failures trigger a tool-guided repair loop. Valid candidates are scored by ADP to update the best RTL or record an unsuccessful strategy, and the process repeats until the query budget is exhausted.

### 4.3 Experience-driven agentic workflow

We implement an experience-driven agentic workflow to mimic the iterative refinement process of hardware engineers, who rely on tool feedback for design correction and continuous experience accumulation. While the population-based search navigates structural innovations through evolutionary mechanisms, this paradigm focuses on utilizing accumulated "analysis experiences" to refine optimization strategies over time. As illustrated in Figure 3, the workflow begins with the original RTL design and an empty experience set, which tracks failed optimization attempts to prevent the model from repeating redundant or suboptimal strategies. In each iteration, the LLM attempts to optimize the current best design using a prompt that incorporates the RTL code, specific design instructions, and accumulated experiences. If a generated variant fails synthesis or logic equivalence checking (LEC), the agent leverages tool feedback to attempt repairs, with a budget of three retries. Once a design is verified as functionally equivalent, the system calculates its ADP to measure improvement. If the design establishes a new performance baseline, it is committed, and the analysis experiences are cleared to reset the optimization context for the next phase of refinement. Conversely, if no improvement is achieved, the attempt is logged as a "bad strategy" to explicitly guide the LLM away from similar implementation paths in future queries. This closed-loop process continues until the query budget is exhausted, ensuring the final output represents the most efficient design discovered through iterative, tool-guided experience management.

## 5 Experiments

### 5.1 Experimental setup

We conduct a comprehensive evaluation across a diverse array of LLMs, encompassing both general-purpose foundation models and domain-specific hardware models.

**Model suite.** The complete model list and grouping are provided in Appendix B; decoding configurations and evaluation cost are reported in Appendices G and H, respectively.

**Toolchain.** The evaluation toolchain and logic-equivalence settings are detailed in Appendices C and I, respectively.

Table 1: Single-shot generation evaluation results. All values are reported in percentages. dark grey indicates the best result, and light grey indicates the second-best result per metric.

| Model | syn@1 | syn@5 | syn@10 | fun@1 | fun@5 | fun@10 | opt@1 | opt@5 | opt@10 | eff@1 | eff@5 | eff@10 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|
| GPT-5.2 | 89.74 | 99.06 | 99.55 | 47.05 | 71.58 | 78.20 | 25.38 | 50.66 | 59.78 | 20.00 | 42.17 | 51.71 |
| GPT-4o | 86.08 | 98.16 | 99.31 | 23.94 | 42.89 | 50.37 | 12.80 | 29.04 | 36.72 | 10.07 | 24.11 | 30.77 |
| GPT-4o-mini | 63.96 | 76.90 | 78.62 | 16.55 | 31.77 | 37.91 | 8.40 | 19.89 | 25.85 | 5.98 | 16.19 | 21.97 |
| Grok-4-1-Fast | 83.42 | 98.45 | 99.47 | 15.17 | 31.89 | 38.67 | 9.32 | 21.76 | 27.48 | 7.95 | 18.99 | 24.48 |
| Grok-4-1-Fast-Reasoning | 86.48 | 97.51 | 98.48 | 63.65 | 85.88 | 90.29 | 38.02 | 68.45 | 77.96 | 31.88 | 60.98 | 72.16 |
| Grok-Code-Fast-1 | 91.04 | 99.33 | 99.61 | 51.67 | 82.87 | 89.90 | 26.35 | 58.46 | 70.91 | 21.43 | 49.69 | 62.29 |
| Llama-3.1-8B | 62.29 | 93.86 | 98.07 | 16.08 | 49.41 | 66.52 | 4.83 | 18.63 | 29.45 | 3.74 | 14.61 | 23.25 |
| Llama-3.1-70B | 82.03 | 98.42 | 99.42 | 22.53 | 53.19 | 64.52 | 11.55 | 32.30 | 43.21 | 9.00 | 26.56 | 36.60 |
| Llama-3.1-405B | 83.19 | 97.99 | 99.12 | 23.33 | 46.38 | 56.22 | 13.40 | 30.13 | 38.47 | 11.05 | 26.00 | 33.92 |
| Llama-4-maverick | 86.79 | 97.59 | 98.33 | 25.16 | 45.59 | 53.04 | 12.00 | 28.96 | 38.17 | 9.21 | 23.65 | 31.83 |
| DeepSeek-Chat | 92.66 | 99.43 | 99.86 | 26.28 | 45.93 | 53.32 | 15.69 | 32.64 | 40.34 | 12.90 | 27.86 | 35.42 |
| DeepSeek-Reasoner | 92.20 | 99.21 | 99.69 | 47.93 | 75.72 | 83.03 | 27.62 | 54.93 | 65.62 | 22.69 | 47.57 | 58.51 |
| DeepSeek-Coder-6.7B | 76.79 | 96.42 | 97.71 | 19.39 | 55.71 | 72.97 | 3.89 | 14.38 | 21.69 | 2.89 | 11.40 | 17.95 |
| CodeLlama-7B-Instruct | 70.68 | 96.07 | 97.94 | 47.55 | 87.75 | 93.75 | 2.48 | 9.99 | 16.78 | 1.67 | 6.79 | 11.73 |
| CodeQwen1.5-7B-Chat | 71.72 | 97.15 | 98.64 | 19.24 | 54.63 | 70.83 | 4.64 | 17.22 | 26.77 | 3.48 | 13.20 | 20.66 |
| CodeV-DS-6.7B | 8.37 | 26.38 | 39.97 | 3.56 | 15.16 | 25.90 | 0.28 | 1.29 | 2.42 | 0.18 | 0.85 | 1.66 |
| CodeV-CL-7B | 18.04 | 58.24 | 78.04 | 12.41 | 44.77 | 64.43 | 0.94 | 4.12 | 7.18 | 0.59 | 2.63 | 4.68 |
| CodeV-QW-7B | 6.42 | 20.00 | 30.90 | 1.89 | 8.43 | 15.02 | 0.40 | 1.82 | 3.24 | 0.17 | 0.84 | 1.61 |
| RTLCoder-Deepseek-v1.1 | 75.90 | 96.18 | 97.34 | 47.78 | 88.22 | 93.97 | 3.13 | 11.84 | 19.20 | 2.12 | 8.22 | 13.59 |

### 5.2 Single-shot generation results

As presented in Table 1, the experimental results indicate a broad trend where most evaluated models successfully generate synthesizable designs, yet performance drops significantly when targeting functional equivalence and optimization. For instance, while DeepSeek-Chat achieves near-perfect syntactic validity, its success rates for functional equivalence and optimization fall sharply. This confirms that while current models can readily generate valid Verilog, generating functionally correct and *efficient* hardware that surpasses existing baselines remains a distinct challenge.

Within this challenging landscape, reasoning-enhanced models significantly outperform their standard counterparts, as evidenced by the substantial performance boost achieved by DeepSeek-Reasoner and Grok-4-1-Fast-Reasoning. Grok-4-1-Fast-Reasoning emerges as the top performer, achieving the highest optimization and efficiency scores. This indicates that reasoning capabilities are essential for RTL optimization. However, a critical gap remains: even the strongest models fail to fully match the optimization targets established by e-graph rewriting. Furthermore, while open-weights models are competitive, they still lag behind top-tier proprietary systems. Given the proprietary nature of hardware design, relying solely on commercial APIs is often infeasible, highlighting the critical need for the community to develop specialized, high-performance open models that can bridge this gap.

Notably, models specifically fine-tuned for Verilog code generation perform poorly on this optimization task. While RTLCoder-Deepseek-v1.1 achieves high functional correctness (highest fun@5 and fun@10), deep analysis suggests it largely acts as an identity function, *i.e.*, preserving the input structure to maintain correctness at the expense of optimization. Other fine-tuned models, such as the CodeV family, struggle across all metrics. These failures indicate that current fine-tuning strategies are insufficient, necessitating new methodologies specifically aimed at teaching models to reason about hardware efficiency rather than merely completing syntax and functionality. Standard errors are reported in Appendix J. We provide detailed per-complexity and per-category results in Appendix K. Selective evaluation results from commercial synthesis tools are reported in Appendix M. Furthermore, a comparative analysis against a specification-only generation baseline is detailed in Appendix N.

### 5.3 Evolutionary search vs. agentic workflow

To enable a fair comparison across different interaction paradigms, we adapt the `syn@k`, `fun@k`, `opt@k`, and `eff@k` metrics into aggregated variants (`syn`, `fun`, `opt`, and `eff`) that emphasize the peak performance attainable within a fixed budget of 20 LLM queries per problem. For both the population-based test-time search and the experience-driven agentic workflow, which involve a single trajectory of up to 20 refinements or mutations, we compute metrics based on the highest-performing variant in the trajectory. For single-shot generation, we apply the same aggregation by selecting the best sample from 20 independent runs rather than estimating pass probabilities via the unbiased

Table 2: Comparison of single-shot generation, population-based test-time search, and experience-driven agentic workflow. All values are reported in percentages. dark grey indicates the best result, and light grey indicates the second-best among the three interaction paradigms.

| Model | Single-Shot Generation | | | | Evolutionary Search | | | | Agentic Workflow | | | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|
| | syn | fun | opt | eff | syn | fun | opt | eff | syn | fun | opt | eff |
| DeepSeek-Chat | 100 | 54.90 | 45.10 | 39.04 | 86.27 | 76.47 | 58.82 | 53.52 | 100 | 90.20 | 56.86 | 44.18 |
| DeepSeek-Reasoner | 100 | 70.59 | 68.63 | 59.89 | 92.16 | 90.20 | 86.27 | 73.68 | 100 | 98.04 | 84.31 | 76.90 |
| Grok-4-1-Fast | 100 | 33.33 | 29.41 | 29.32 | 86.27 | 72.55 | 54.90 | 45.03 | 92.16 | 74.51 | 56.86 | 47.74 |
| Grok-4-1-Fast-Reasoning | 100 | 88.24 | 84.31 | 80.98 | 88.24 | 80.39 | 74.51 | 68.03 | 98.04 | 88.24 | 86.27 | 81.18 |
| Grok-Code-Fast-1 | 99.99 | 88.24 | 80.39 | 67.72 | 90.20 | 86.27 | 68.63 | 61.78 | 100 | 98.04 | 76.47 | 68.11 |
| GPT-4o-mini | 90.20 | 39.22 | 35.29 | 35.29 | 84.31 | 64.71 | 47.06 | 37.37 | 98.04 | 52.94 | 41.18 | 38.88 |
| GPT-4o | 98.04 | 56.86 | 50.98 | 43.56 | 86.27 | 80.39 | 52.94 | 46.10 | 94.12 | 70.59 | 54.90 | 48.81 |
| GPT-5.2 | 98.04 | 76.47 | 70.59 | 56.03 | 90.20 | 76.47 | 70.59 | 59.95 | 100 | 100 | 88.24 | 82.35 |
| GPT-5.5 | 100.00 | 92.16 | 62.75 | 55.47 | 84.31 | 82.35 | 60.78 | 57.48 | 98.04 | 94.12 | 78.43 | 75.55 |
| Claude-Opus-4.8 | 98.04 | 86.27 | 54.90 | 52.32 | 86.27 | 84.31 | 64.71 | 63.22 | 96.08 | 94.12 | 68.63 | 67.39 |

estimator. This ensures all paradigms are evaluated under equivalent query budgets, focusing on the peak performance attainable rather than cumulative success rates.

Table 2 reveals that both the population-based test-time search and experience-driven agentic workflow significantly outperform single-shot generation across most metrics, highlighting the benefits of parallel evolutionary exploration and tool-driven iterative refinement. However, a paradoxical trend emerges: the population-based test-time search consistently underperforms in syntactic correctness (`syn`) compared to independent single-shot attempts. This suggests that without granular diagnostic feedback, mutating existing RTL code often risks propagating or compounding syntactic errors. Consequently, when syntax error information is sparse, evolving a flawed design is less effective than regenerating it from scratch. While evolutionary search excels in deeper architectural optimization, simple error recovery may be more efficiently handled via diverse sampling of standalone models.

The population-based test-time search generally demonstrates suboptimal performance relative to the experience-driven agentic workflow during optimization due to three primary factors. First, the efficacy of the evolutionary search is fundamentally limited; it lacks robust, sequential refinement mechanisms to ensure preservation of functional equivalence during complex mutations compared to the agentic workflow. Second, the informational granularity recorded during the evolutionary search is notably restricted; it typically records only modified code snippets and their corresponding fitness scores, whereas the experience-driven agentic workflow archives richer contextual data, such as analytical rationales for each modification, to provide more actionable insights. Finally, while the population-based test-time search functions as a closed-loop system, the agentic workflow utilizes enriched prompts with specific instructions that actively steer the LLM toward effective RTL optimization strategies. For highly capable models like Grok4-1-Fast-Reasoning and GPT-5.2, generic evolutionary search yields only marginal improvements. To unlock significant performance gains, it is imperative to design domain-specific components, including customized workflows, optimized prompts, and structured memory mechanisms.

## 6 Conclusion

As LLMs become increasingly integrated into the hardware design workflow, the focus must shift beyond simple code generation to the rigorous engineering of efficient hardware. We address the critical bottleneck of RTL optimization by introducing **TurboRTL**, the first large-scale benchmark that utilizes e-graph rewriting within an automated pipeline to systematically identify designs with non-trivial optimization potential. Moving beyond the limitations of single-shot evaluations, our framework incorporates two advanced, tool-augmented interaction paradigms—population-based test-time search and experience-driven agentic workflows—to accurately mirror the iterative refinement process employed by hardware engineers. Our extensive evaluation demonstrates that while individual LLMs show initial promise, advanced agentic strategies are essential to tackle the complexities of this domain. By providing both a high-quality benchmark and a diverse evaluation pipeline, this work establishes a critical foundation for advancing the frontier of automated RTL optimization.

## Ethics Statement

We have reviewed the ICLR Code of Ethics[1] and find no substantial ethical concerns in this work. We use RTL designs from open-source datasets in accordance with their licenses, and the study involves no human participants or private personal data.

## Reproducibility Statement

The paper documents the benchmark construction, synthesis and equivalence checks, evaluation protocols, interaction paradigms, and metrics in Sections 3–5 and the appendix. The supplementary material provides the benchmark data, code, configurations, and reproduction scripts needed to repeat the reported experiments. The data and code are also available in our anonymous repository: https://anonymous.4open.science/r/turbortl-67C7.

## AI Usage Statement

Generative AI tools were used for limited manuscript drafting and language editing, and as auxiliary tools for parts of the benchmark annotation and specification preparation. Their outputs were reviewed by the authors and domain experts. The authors remain responsible for all claims, data, code, and results.

## References

Manar Abdelatty, Maryam Nouh, Jacob K Rosenstein, and Sherief Reda. Pluto: A benchmark for evaluating efficiency of llm-generated hardware code. *arXiv preprint arXiv:2510.14756*, 2025.

Anthropic. Claude opus 4.8 model documentation. Technical report, Anthropic, 2026.

Anonymous Authors. RuleNet: Learning design-specific rule subsets for efficient RTL optimization. Under review at AAAI, 2027. Filename: RuleNet_AAAI_2027_Anonymized.pdf.

Jinze Bai, Shuai Bai, Yunfei Chu, Zeyu Cui, Kai Dang, Xiaodong Deng, Yang Fan, Wenbin Ge, Yu Han, Fei Huang, et al. Qwen technical report. *arXiv preprint arXiv:2309.16609*, 2023.

Robert Brayton and Alan Mishchenko. Abc: An academic industrial-strength verification tool. In *International Conference on Computer Aided Verification*, pp. 24–40. Springer, 2010.

Lei Chen, Yiqi Chen, Zhufei Chu, Wenji Fang, Tsung-Yi Ho, Ru Huang, Yu Huang, Sadaf Khan, Min Li, Xingquan Li, et al. Large circuit models: opportunities and challenges. *Science China Information Sciences*, 67(10):200402, 2024.

Mark Chen, Jerry Tworek, Heewoo Jun, Qiming Yuan, Henrique Ponde de Oliveira Pinto, Jared Kaplan, Harri Edwards, Yuri Burda, Nicholas Joseph, Greg Brockman, Alex Ray, Raul Puri, Gretchen Krueger, Michael Petrov, Heidy Khlaaf, Girish Sastry, Pamela Mishkin, Brooke Chan, Scott Gray, Nick Ryder, Mikhail Pavlov, Alethea Power, Lukasz Kaiser, Mohammad Bavarian, Clemens Winter, Philippe Tillet, Felipe Petroski Such, Dave Cummings, Matthias Plappert, Fotios Chantzis, Elizabeth Barnes, Ariel Herbert-Voss, William Hebgen Guss, Alex Nichol, Alex Paino, Nikolas Tezak, Jie Tang, Igor Babuschkin, Suchir Balaji, Shantanu Jain, William Saunders, Christopher Hesse, Andrew N. Carr, Jan Leike, Josh Achiam, Vedant Misra, Evan Morikawa, Alec Radford, Matthew Knight, Miles Brundage, Mira Murati, Katie Mayer, Peter Welinder, Bob McGrew, Dario Amodei, Sam McCandlish, Ilya Sutskever, and Wojciech Zaremba. Evaluating large language models trained on code. *arXiv preprint arXiv:2107.03374*, 2021.

Xiangli Chen, Yuehua Meng, and Gang Chen. Incremental verilog parser. In *2023 International Symposium of Electronics Design Automation (ISEDA)*, pp. 236–240. IEEE, 2023.

[1] https://iclr.cc/public/CodeOfEthics

Samuel Coward, Theo Drane, and George A Constantinides. Rover: Rtl optimization via verified e-graph rewriting. *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, 43(12):4687–4700, 2024.

Google DeepMind. Gemini 3 technical report. Technical report, Google, 2025. URL https://deepmind.google/technologies/gemini/.

Aaron Grattafiori, Abhimanyu Dubey, Abhinav Jauhri, Abhinav Pandey, Abhishek Kadian, Ahmad Al-Dahle, Aiesha Letman, Akhil Mathur, Alan Schelten, Alex Vaughan, et al. The llama 3 herd of models. *arXiv preprint arXiv:2407.21783*, 2024.

Daya Guo, Qihao Zhu, Dejian Yang, Zhenda Xie, Kai Dong, Wentao Zhang, Guanting Chen, Xiao Bi, Yu Wu, YK Li, et al. Deepseek-coder: When the large language model meets programming–the rise of code intelligence. *arXiv preprint arXiv:2401.14196*, 2024.

Daya Guo, Dejian Yang, Haowei Zhang, Junxiao Song, Peiyi Wang, Qihao Zhu, Runxin Xu, Ruoyu Zhang, Shirong Ma, Xiao Bi, et al. Deepseek-r1 incentivizes reasoning in llms through reinforcement learning. *Nature*, 645(8081):633–638, 2025.

Chia-Tung Ho, Haoxing Ren, and Brucek Khailany. Verilogcoder: Autonomous verilog coding agents with graph-based planning and abstract syntax tree (ast)-based waveform tracing tool. In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 39, pp. 300–307, 2025.

Dorit S Hochbaum and David B Shmoys. A best possible heuristic for the k-center problem. *Mathematics of operations research*, 10(2):180–184, 1985.

Aaron Hurst, Adam Lerer, Adam P Goucher, Adam Perelman, Aditya Ramesh, Aidan Clark, AJ Ostrow, Akila Welihinda, Alan Hayes, Alec Radford, et al. Gpt-4o system card. *arXiv preprint arXiv:2410.21276*, 2024.

IEEE Computer Society. *IEEE Standard for Verilog Hardware Description Language*, 2006. IEEE Std 1364-2005 (Revision of IEEE Std 1364-2001).

Zeju Li, Changran Xu, Zhengyuan Shi, Zedong Peng, Yi Liu, Yunhao Zhou, Lingfeng Zhou, Chengyu Ma, Jianyuan Zhong, Xi Wang, et al. Deepcircuitx: A comprehensive repository-level dataset for rtl code understanding, generation, and ppa analysis. In *2025 IEEE International Conference on LLM-Aided Design (ICLAD)*, pp. 204–211. IEEE, 2025.

Mingjie Liu, Teodor-Dumitru Ene, Robert Kirby, Chris Cheng, Nathaniel Pinckney, Rongjian Liang, Jonah Alben, Himyanshu Anand, Sanmitra Banerjee, Ismet Bayraktaroglu, et al. Chipnemo: Domain-adapted llms for chip design. *arXiv preprint arXiv:2311.00176*, 2023a.

Mingjie Liu, Nathaniel Pinckney, Brucek Khailany, and Haoxing Ren. Verilogeval: Evaluating large language models for verilog code generation. In *2023 IEEE/ACM International Conference on Computer Aided Design (ICCAD)*, pp. 1–8. IEEE, 2023b.

Mingjie Liu, Yun-Da Tsai, Wenfei Zhou, and Haoxing Ren. Craftrtl: High-quality synthetic data generation for verilog code models with correct-by-construction non-textual representations and targeted code repair. In *The Thirteenth International Conference on Learning Representations*, 2025a.

Shang Liu, Wenji Fang, Yao Lu, Jing Wang, Qijun Zhang, Hongce Zhang, and Zhiyao Xie. Rtlcoder: Fully open-source and efficient llm-assisted rtl code generation technique. *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, 2024.

Yi Liu, XU Changran, Yunhao Zhou, Zeju Li, and Qiang Xu. Deeprtl: Bridging verilog understanding and generation with a unified representation model. In *The Thirteenth International Conference on Learning Representations*, 2025b.

Yi Liu, Hongji Zhang, Yunhao Zhou, Zhengyuan Shi, Changran Xu, and Qiang Xu. Deeprtl2: A versatile model for rtl-related tasks. In *Findings of the Association for Computational Linguistics: ACL 2025*, 2025c.

Yao Lu, Shang Liu, Qijun Zhang, and Zhiyao Xie. Rtllm: An open-source benchmark for design rtl generation with large language model. In *2024 29th Asia and South Pacific Design Automation Conference (ASP-DAC)*, pp. 722–727. IEEE, 2024.

Yao Lu, Shang Liu, Hangan Zhou, Wenji Fang, Qijun Zhang, and Zhiyao Xie. A new benchmark for the appropriate evaluation of rtl code optimization. *arXiv preprint arXiv:2601.01765*, 2026.

Leland McInnes, John Healy, Steve Astels, et al. hdbscan: Hierarchical density based clustering. *J. Open Source Softw.*, 2(11):205, 2017.

Meta. Llama 4 technical report. Technical report, Meta, 2025. URL https://ai.meta.com/blog/llama-4-multimodal-intelligence/.

Arvind Neelakantan, Tao Xu, Raul Puri, Alec Radford, Jesse Michael Han, Jerry Tworek, Qiming Yuan, Nikolas Tezak, Jong Wook Kim, Chris Hallacy, et al. Text and code embeddings by contrastive pre-training. *arXiv preprint arXiv:2201.10005*, 2022.

Alexander Novikov, Ngân Vũ, Marvin Eisenberger, Emilien Dupont, Po-Sen Huang, Adam Zsolt Wagner, Sergey Shirobokov, Borislav Kozlovskii, Francisco JR Ruiz, Abbas Mehrabian, et al. Alphaevolve: A coding agent for scientific and algorithmic discovery. *arXiv preprint arXiv:2506.13131*, 2025.

OpenAI. tiktoken: fast bpe tokeniser for use with openai's models. https://github.com/openai/tiktoken, 2023.

OpenAI. Gpt-5.2 technical report. Technical report, OpenAI, 2025. URL https://openai.com/index/introducing-gpt-5-2/.

OpenAI. Gpt-5.5 model documentation. Technical report, OpenAI, 2026.

Zehua Pei, Hui-Ling Zhen, Mingxuan Yuan, Yu Huang, and Bei Yu. Betterv: controlled verilog generation with discriminative guidance. In *Proceedings of the 41st International Conference on Machine Learning*, pp. 40145–40153, 2024.

Ruizhong Qiu, Weiliang Will Zeng, James Ezick, Christopher Lott, and Hanghang Tong. How efficient is llm-generated code? a rigorous & high-standard benchmark. In *The Thirteenth International Conference on Learning Representations*, 2025.

Baptiste Roziere, Jonas Gehring, Fabian Gloeckle, Sten Sootla, Itai Gat, Xiaoqing Ellen Tan, Yossi Adi, Jingyu Liu, Romain Sauvestre, Tal Remez, et al. Code llama: Open foundation models for code. *arXiv preprint arXiv:2308.12950*, 2023.

Ross Tate, Michael Stepp, Zachary Tatlock, and Sorin Lerner. Equality saturation: a new approach to optimization. In *Proceedings of the 36th annual ACM SIGPLAN-SIGACT symposium on Principles of programming languages*, pp. 264–276, 2009.

Shailja Thakur, Baleegh Ahmad, Hammond Pearce, Benjamin Tan, Brendan Dolan-Gavitt, Ramesh Karri, and Siddharth Garg. Verigen: A large language model for verilog code generation. *ACM Transactions on Design Automation of Electronic Systems*, 29(3):1–31, 2024.

Max Willsey, Chandrakana Nandi, Yisu Remy Wang, Oliver Flatt, Zachary Tatlock, and Pavel Panchekha. Egg: Fast and extensible equality saturation. *Proceedings of the ACM on Programming Languages*, 5(POPL):1–29, 2021.

Clifford Wolf, Johann Glaser, and Johannes Kepler. Yosys-a free verilog synthesis suite. In *Proceedings of the 21st Austrian Workshop on Microelectronics (Austrochip)*, volume 97, 2013.

xAI. Grok 4.1 fast technical report. Technical report, xAI, 2025a. URL https://x.ai/news/grok-4-1-fast.

xAI. Grok code fast 1 technical report. Technical report, xAI, 2025b. URL https://x.ai/news/grok-code-fast-1.

Changran Xu, Yi Liu, Yunhao Zhou, Shan Huang, Ningyi Xu, and Qiang Xu. Speculative decoding for verilog: Speed and quality, all in one. *The 62nd ACM/IEEE Design Automation Conference (DAC)*, 2025.

Ziqi Yan, Jiqiang Liu, Gang Li, Zhen Han, and Shuo Qiu. Privmin: Differentially private minhash for jaccard similarity computation. *arXiv preprint arXiv:1705.07258*, 2017.

Xufeng Yao, Yiwen Wang, Xing Li, Yingzhao Lian, Ran Chen, Lei Chen, Mingxuan Yuan, Hong Xu, and Bei Yu. Rtlrewriter: Methodologies for large models aided rtl code optimization. In *Proceedings of the 43rd IEEE/ACM International Conference on Computer-Aided Design*, pp. 1–7, 2024.

Yongan Zhang, Zhongzhi Yu, Yonggan Fu, Cheng Wan, and Yingyan Celine Lin. Mg-verilog: Multi-grained dataset towards enhanced llm-assisted verilog generation. In *2024 IEEE LLM Aided Design Workshop (LAD)*, pp. 1–5. IEEE, 2024.

Yang Zhao, Di Huang, Chongxiao Li, Pengwei Jin, Muxin Song, Yinan Xu, Ziyuan Nan, Mingju Gao, Tianyun Ma, Lei Qi, et al. Codev: Empowering llms with hdl generation through multi-level summarization. *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, 2025.

## A LIMITATIONS

We identify three primary limitations in this work. First, TurboRTL currently focuses on single-module designs, whereas industrial hardware typically comprises large-scale, hierarchical multi-module systems. However, our dataset construction pipeline and evaluation framework are inherently extensible, and we aim to scale the benchmark to include multi-module designs in future work. Second, resource constraints limit our evaluation scope, and we restrict the evaluation of our population-based test-time search and experience-driven agentic workflows to a subset of the benchmark. We plan to address these coverage gaps in future iterations as resources permit. Third, the egg-based filtering pipeline used during benchmark construction does not guarantee that all designs with optimization potential are identified: some designs may contain meaningful optimization opportunities that egg cannot effectively exploit. Nevertheless, our contribution is a new data-selection methodology that identifies a high-value subset with validated optimization potential, ultimately yielding the 288-design benchmark. Future work will investigate complementary selection strategies to broaden coverage beyond the designs captured by the current egg-based pipeline.

## B EVALUATED MODELS

The evaluated models are categorized into four tiers. First, our proprietary state-of-the-art (SOTA) selection includes GPT-5.2 (OpenAI, 2025), GPT-4o, and GPT-4o-mini (Hurst et al., 2024), alongside Grok-4-1-Fast and its reasoning variant (xAI, 2025a), and Grok-Code-Fast-1 (xAI, 2025b). Second, we assess open-weight SOTA models, covering the Llama-3.1 series (8B, 70B, 405B) (Grattafiori et al., 2024), Llama-4-maverick (Meta, 2025), DeepSeek-Chat, and DeepSeek-Reasoner (Guo et al., 2025). Third, we examine general-purpose code models, including DeepSeek-Coder-6.7B (Guo et al., 2024), CodeLlama-7B-Instruct (Roziere et al., 2023), and CodeQwen1.5-7B-Chat (Bai et al., 2023). Finally, we evaluate hardware-specialized models fine-tuned for Verilog generation, including the CodeV family (DS-6.7B, CL-7B, QW-7B) (Zhao et al., 2025) and RTLCoder-Deepseek-v1.1 (Liu et al., 2024).

For the three-way generation-paradigm comparison in Table 2, we additionally evaluate GPT-5.5 (OpenAI, 2026) and Claude-Opus-4.8 (Anthropic, 2026). These broadly capable frontier models provide a stronger and more comprehensive test of TurboRTL, while their remaining performance gap illustrates that the benchmark remains challenging even for advanced general-purpose systems.

## C EDA TOOLCHAIN

To facilitate accessibility and reproducibility, the primary evaluation pipeline uses open-source EDA tools. We use Yosys (Wolf et al., 2013) for logic synthesis and ABC (Brayton & Mishchenko, 2010) for technology mapping and quality estimation. This open-source flow preserves the validity of the optimization targets established by the EqSat-driven filtering process. For robustness, we also report selective evaluations with the commercial Synopsys Design Compiler (DC) flow and use them to assess cross-toolchain consistency. The detailed logic-equivalence checking configuration is provided in Appendix I.

## D CHOICE OF OPTIMIZATION OBJECTIVE

In this work, we restrict the objective of the RTL optimization to area and timing metrics (*i.e.*, ADP calculated from area and delay), rather than treating power as a first-class optimization target. The key reason is that total power consists of static power (*leakage*) and dynamic power (*switching/internal*), where dynamic power is strongly coupled to the signal switching activity factor $\alpha$. A common approximation is

$$P_{\text{dyn}} \propto \alpha C V^2 f \tag{5}$$

where $C$ is the effective load capacitance, $V$ is the supply voltage, and $f$ is the clock frequency. Since $\alpha$ is determined by real input distributions, control flow, and clock-enable/gating behavior, a methodologically sound dynamic-power evaluation requires representative stimuli and trustworthy activity information.

Reliable power analysis is typically activity-driven and requires simulation-generated activity annotations such as SAIF (*Switching Activity Interchange Format*) or VCD/FSDB (*Value Change Dump*/waveforms) under representative input stimuli or real workloads. SAIF summarizes toggle counts and state residency, while VCD/FSDB captures time-resolved transitions; both enable realistic switching-activity annotation for gate-level power models. In our experimental setting, such representative stimuli and SAIF/VCD are unavailable. As a result, synthesis tools (*e.g.*, Synopsys Design Compiler) must rely on default or assumed toggle-rate/static-probability models, yielding power estimates that are highly sensitive to workload distribution, clock-enable/gating behavior, and state coverage, and thus unreliable for comparing design points.

By contrast, area and timing are largely determined by the synthesized mapping, the technology library, and timing constraints, and are substantially less sensitive to missing activity information.

## E E-GRAPH REWRITING DETAILS

This section provides a detailed description of the e-graph rewriting process, specifically addressing the technical challenges of Verilog parsing, sequential circuit handling, and optimal variant extraction. To ensure robust parsing and broad coverage, we utilize Stagira (Chen et al., 2023), a formally verified Verilog parser compliant with the IEEE 1364-2005 standard (IEEE Computer Society, 2006). This compliance ensures that a wide array of Verilog syntax is accurately translated into S-expressions for e-graph rewriting, maintaining the syntactic integrity of the original design during its transformation into a symbolic prefix-form expression.

Regarding the handling of sequential circuits, the current e-graph rewriting rules focus exclusively on algebraic and control logic transformations occurring between registers. By deliberately omitting structural rewrites of sequential elements (*e.g.*, re-timing or finite state machine state reduction), we ensure that all generated designs remain cycle-accurate and preserve original timing boundaries. This approach simplifies the complexity of sequential equivalence checking while still enabling significant structural and logic optimization within the combinational paths of the RTL.

Furthermore, acknowledging that heuristic searches within the `egg` library cannot strictly guarantee a global optimum during extraction, we adopt a multi-candidate extraction strategy. We extract multiple distinct variants across different e-graph rewriting iterations. These candidates are subsequently processed through a rigorous synthesis flow using Synopsys Design Compiler with the `compile_ultra` command. By characterizing these variants against the SkyWater 130 nm technology library, we select the final variant that demonstrates the most superior ADP metric, ensuring that the benchmark samples reflect validated, non-trivial optimization potential.

## F QUANTITATIVE COMPARISON WITH PRIOR WORK

To demonstrate the advancement of the TurboRTL benchmark within the existing evaluation landscape, we provide a comprehensive quantitative comparison against prior RTL optimization benchmarks, including RTL-OPT (Lu et al., 2026), Pluto (Abdelatty et al., 2025), and RTLRewriter (Yao et al., 2024). Since Pluto itself is not open-sourced, we conduct our comparisons using the RTLLM (Lu et al., 2024) and VerilogEval (Human and Machine) (Liu et al., 2023b) datasets from which Pluto's samples are sourced. We evaluate these benchmarks across five complementary dimensions: dataset scale, design complexity, optimization potential, semantic coverage, and structural diversity.

Table 3: Dataset Scale, Complexity, and Optimization Potential

| Benchmark | # Designs | Avg. Lines | Avg. Tokens | Avg. Complexity | Opt. Potential |
|---|---|---|---|---|---|
| RTL-OPT | 36 | 31.45 | 277.73 | 0.28 | Human |
| RTLLM-v2.0 | 50 | 54.72 | 415.20 | 0.80 | Human |
| RTLRewriter | 76 | 67.49 | 657.37 | 0.76 | Artificial Noise |
| VerilogEval-H | 156 | 19.60 | 119.15 | 0.52 | Human |
| VerilogEval-M | 143 | 17.36 | 95.98 | 0.42 | Human |
| **TurboRTL** | **288** | **129.48** | **1252.34** | **0.91** | **EqSat-Driven** |

**Dataset scale, complexity, and optimization potential.** Unlike prior benchmarks that rely on manual human curation or the insertion of artificial noise, TurboRTL leverages an automated, EqSat-driven pipeline to identify designs with significant optimization potential from a massive corpus. Table 3 highlights that TurboRTL significantly exceeds previous works in mean line counts, token counts (calculated using the `tiktoken` tokenizer (OpenAI, 2023)), and average design complexity (defined in Appendices K and L). While benchmarks like Pluto and RTL-OPT utilize human effort to ensure optimization potential, they often suffer from scalability and diversity bottlenecks. Conversely, RTLRewriter relies on artificial noise (*e.g.*, redundant wire assignments) that commercial tools can often eliminate trivially. By contrast, TurboRTL ensures every sample possesses a mathematically proven path to enhanced efficiency that remains non-trivial for state-of-the-art commercial compilers.

Table 4: Semantic Coverage

| Benchmark | # Cats | # Subcats | Top-3 Cat. Frac. | Top-5 Sub. Frac. |
|---|---|---|---|---|
| RTL-OPT | 4 | 22 | 0.975 | 0.475 |
| RTLLM-v2.0 | 8 | 35 | 0.720 | 0.320 |
| RTLRewriter | 7 | 48 | 0.684 | 0.303 |
| VerilogEval-H | 7 | 58 | 0.801 | 0.378 |
| VerilogEval-M | 7 | 57 | 0.818 | 0.385 |
| **TurboRTL** | **9** | **115** | **0.639** | **0.236** |

**Semantic coverage.** A critical requirement for a good RTL optimization benchmark is the ability to represent a broad spectrum of hardware design functionalities. We quantify this using semantic coverage, measured by the count of categories and subcategories (obtained with Gemini-3-pro), as well as the concentration of the most frequent design types. As shown in Table 4, TurboRTL spans the largest number of categories and subcategories. Furthermore, its lower top-$k$ concentration metrics indicate that the dataset is less dominated by repetitive design families compared to prior benchmarks, which often exhibit high redundancy in simple logic structures.

Table 5: Structural Diversity

| Benchmark | # Signatures | Feat. Coverage | Pairwise Dist. | Novelty vs. Others |
|---|---|---|---|---|
| RTL-OPT | 37 | 0.688 | 0.146 | 0.015 |
| RTLLM-v2.0 | 47 | 0.772 | 0.183 | 0.029 |
| RTLRewriter | 66 | 0.811 | 0.210 | 0.037 |
| VerilogEval-H | 87 | 0.650 | 0.135 | 0.003 |
| VerilogEval-M | 74 | 0.600 | 0.124 | 0.001 |
| **TurboRTL** | **223** | **0.976** | **0.213** | **0.061** |

**Structural diversity.** We further evaluate the benchmark's structural diversity by measuring "feature coverage", which assesses how much of a fixed structural-feature vocabulary (*e.g.*, FSM size, pipeline depth, arithmetic/control intensity) is exercised, and "novelty", which quantifies the average nearest-neighbor distance to designs in other benchmarks. As detailed in Table 5, TurboRTL achieves the highest overall structural coverage (0.976) and the greatest within-benchmark spread (0.213). These metrics demonstrate that TurboRTL provides a more diverse and challenging testbed for navigating complex architectural trade-offs in hardware design optimization.

**Broader impact.** We view TurboRTL as a necessary foundation for bridging the gap between academic benchmarks and industrial production. By shifting the objective from mere syntax generation to structural optimization—a primary bottleneck in modern hardware design—we expose a fundamental capability gap in current state-of-the-art models. Even the strongest reasoning-enhanced LLMs often fail to match the optimization targets established by our automated e-graph pipeline. Mastering this "local" optimization at the single-module level is a prerequisite for the eventual "global" optimization required for hierarchical, multi-module industrial designs. TurboRTL thus establishes the rigorous engineering standard required to bring AI-assisted hardware design into practical application.

## G DECODING DETAILS

To ensure a fair comparison across the 19 diverse models in our evaluation suite, we employ a uniform decoding configuration. A temperature of 0.7 is applied across all models; for local open-weight models, we additionally utilize nucleus sampling with top-p = 0.95. Reasoning behavior is controlled by the specific model variant selected (*e.g.*, reasoning vs. non-reasoning versions) rather than through a separate reasoning-effort parameter. We prioritize this fixed setting to maintain a stable baseline across the benchmark, with specific per-model output-length limits and defaults strictly observed.

## H EVALUATION COST AND LATENCY

The evaluation of LLMs on RTL optimization tasks involves significant computational overhead and synthesis delays, which we quantify here using GPT-4o as a representative example. In a single-shot generation setting with 20 queries per design, evaluating the 288-design TurboRTL suite incurs a total API cost of $111.68 and a sequential LLM latency of 13.43 hours. The subsequent hardware synthesis and functional verification process requires an additional 38.8 hours on a high-performance workstation equipped with Intel Xeon Platinum 8480C CPUs (112 cores) and 500GB RAM. These substantial resource requirements justify our strategic decision to curate the representative 288-design final subset from the 3,414 EqSat-validated candidates, ensuring that the benchmark remains accessible to the research community while highlighting the improvement of evaluation efficiency as a vital direction for future research.

## I LOGIC EQUIVALENCE CHECKING SETTINGS

Our formal verification procedure ensures strict functional preservation between the original RTL and its optimized variants through a combination of combinational and sequential verification techniques. For designs involving purely combinational logic, we utilize the `cec` command from the ABC tool (Brayton & Mishchenko, 2010). For more complex sequential circuits where state transitions must be preserved, we employ ABC's built-in `sec` operator, which is specifically engineered for high-assurance sequential equivalence checking (SEC).

To ensure deterministic verification of sequential designs, we adhere to specific protocols for clock and reset signal handling. We utilize ABC's default configurations, which support designs within a single clock domain with aligned clocks. Crucially, all registers are required to possess dedicated reset ports to ensure consistent initialization across the verification process. For initialization, we employ a standard assumption where all registers are initialized to zero, simplifying the formal setup while remaining aligned with typical hardware synthesis outcomes.

The complexity of the formal proof is managed by tuning the induction depth. In our implementation, we set the induction depth to 4 (typically ranging from 2 to 10), which strikes a balance between ensuring successful proof convergence and avoiding SAT-solver memory explosion. Furthermore, our optimization strategy, which focuses primarily on the logic between registers (as in Appendix E), helps manage scalability by preventing an exponential explosion of the reachable state space. This approach enables the efficient formal verification of complex designs that would otherwise be computationally prohibitive for traditional sequential verification methods.

## J STANDARD ERRORS

To provide a rigorous statistical foundation for the reported metrics, this section details the variability in model performance across the 288 design problems comprising the TurboRTL benchmark. We define the standard error for each metric as the standard deviation of the problem-level success rates divided by the square root of the total number of problems. Specifically, the standard error of the mean (SEM) is calculated as

$$SEM = \sigma_{\text{problems}}/\sqrt{n} \tag{6}$$

where $\sigma_{\text{problems}}$ represents the variability in success rates across the dataset. For metrics involving multiple samples, such as `fun@5` or `opt@10`, the success rate for each individual problem is first

Table 6: Standard error of the mean (SEM) for single-shot evaluation. All values are reported in percentages and reflect the variability across the 288 benchmark problems.

| Model | syn@1 | syn@5 | syn@10 | fun@1 | fun@5 | fun@10 | opt@1 | opt@5 | opt@10 | eff@1 | eff@5 | eff@10 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|
| GPT-5.2 | 1.00 | 0.41 | 0.36 | 2.16 | 2.28 | 2.23 | 1.75 | 2.47 | 2.63 | 1.55 | 2.28 | 2.51 |
| GPT-4o | 1.22 | 0.54 | 0.37 | 1.88 | 2.54 | 2.70 | 1.32 | 2.25 | 2.55 | 1.12 | 2.06 | 2.38 |
| GPT-4o-mini | 2.33 | 2.38 | 2.38 | 1.63 | 2.41 | 2.67 | 1.09 | 1.97 | 2.34 | 0.80 | 1.70 | 2.13 |
| Grok-4-1-Fast | 1.19 | 0.47 | 0.35 | 1.46 | 2.38 | 2.64 | 1.13 | 2.08 | 2.40 | 1.04 | 1.93 | 2.27 |
| Grok-4-1-Fast-Res | 1.29 | 0.75 | 0.65 | 2.02 | 1.71 | 1.58 | 1.90 | 2.19 | 2.12 | 1.80 | 2.17 | 2.15 |
| Grok-Code-Fast-1 | 0.86 | 0.38 | 0.35 | 1.87 | 1.70 | 1.53 | 1.54 | 2.22 | 2.28 | 1.41 | 2.16 | 2.31 |
| Llama-3.1-8B | 1.45 | 0.87 | 0.51 | 0.91 | 1.97 | 2.17 | 0.48 | 1.53 | 2.14 | 0.40 | 1.34 | 1.91 |
| Llama-3.1-70B | 1.11 | 0.50 | 0.27 | 1.38 | 2.33 | 2.48 | 1.03 | 2.14 | 2.53 | 0.88 | 1.93 | 2.38 |
| Llama-3.1-405B | 1.22 | 0.59 | 0.43 | 1.73 | 2.43 | 2.58 | 1.34 | 2.24 | 2.52 | 1.19 | 2.08 | 2.40 |
| Llama-4-maverick | 1.22 | 0.78 | 0.67 | 1.86 | 2.56 | 2.70 | 1.24 | 2.15 | 2.52 | 1.01 | 1.94 | 2.32 |
| DeepSeek-Chat | 0.85 | 0.26 | 0.09 | 1.95 | 2.57 | 2.70 | 1.53 | 2.35 | 2.62 | 1.38 | 2.15 | 2.46 |
| DeepSeek-Reasoner | 0.87 | 0.37 | 0.20 | 2.05 | 2.08 | 1.95 | 1.79 | 2.37 | 2.44 | 1.62 | 2.27 | 2.42 |
| DS-Coder-6.7B | 1.32 | 0.90 | 0.81 | 1.03 | 1.92 | 2.03 | 0.47 | 1.51 | 2.08 | 0.36 | 1.28 | 1.87 |
| CodeLlama-7B-Ins | 1.30 | 0.87 | 0.70 | 1.34 | 1.39 | 1.26 | 0.36 | 1.14 | 1.74 | 0.29 | 0.87 | 1.39 |
| CodeQwen1.5-7B | 1.21 | 0.71 | 0.58 | 1.03 | 2.01 | 2.12 | 0.51 | 1.54 | 2.15 | 0.41 | 1.33 | 1.89 |
| CodeV-DS-6.7B | 0.89 | 1.75 | 2.26 | 0.34 | 1.26 | 1.95 | 0.09 | 0.38 | 0.67 | 0.05 | 0.26 | 0.51 |
| CodeV-CL-7B | 0.77 | 1.68 | 1.78 | 0.61 | 1.76 | 2.17 | 0.18 | 0.75 | 1.21 | 0.13 | 0.54 | 0.91 |
| CodeV-QW-7B | 0.82 | 1.62 | 2.13 | 0.24 | 0.97 | 1.59 | 0.11 | 0.49 | 0.84 | 0.06 | 0.28 | 0.53 |
| RTLCoder-DS-v1.1 | 1.27 | 0.97 | 0.88 | 1.33 | 1.37 | 1.25 | 0.43 | 1.29 | 1.86 | 0.33 | 1.02 | 1.52 |

determined using the unbiased estimator, and the SEM is subsequently derived from the distribution of these rates across the entire suite.

The reported SEM values in Table 6 offer several insights into the stability and difficulty distribution of current LLMs in hardware optimization tasks. The variability for synthesizability (`syn@k`) remains consistently low, often below 1.0% for high-performing models, indicating that the ability to generate syntactically valid Verilog is a robust trait across varying design complexities. In contrast, significantly higher SEM values are observed in the functionality (`fun`) and optimization (`opt`) categories, typically ranging between 1.5% and 2.7%. This increased variance reflects the inherent diversity of the TurboRTL dataset, where specific design categories or high-complexity modules present steeper challenges than others, resulting in a wider distribution of success rates. Notably, the relatively low SEM observed for specialized hardware models in optimization categories is primarily a result of a "floor effect", where these models consistently fail to optimize the majority of the problems, leading to a narrow distribution of near-zero results.

## K Per-complexity and per-category performance analysis

To provide a granular view of model capabilities, we present comprehensive performance breakdowns for each evaluated model in Table 10 through Table 28. These tables show performance across distinct functional categories—including arithmetic, dsp, communication, miscellaneous, datapath, memory, control, processor, and crypto—and varying degrees of design complexity. The detailed criteria used to stratify designs into complexity levels ($C = 0$ to $C = 5$) are defined in Appendix L. Based on these criteria, the benchmark samples in TurboRTL fall within levels 0 to 3. As noted in the Limitations (Appendix A), levels 4 and 5 correspond to large-scale subsystems and industrial-grade integration, which are beyond the current scope of our single-module designs. To ensure classification accuracy, both complexity scores and functional categories are initially analyzed by Gemini-3-pro (Google DeepMind, 2025) and subsequently verified by professional hardware engineers.

**Complexity trends.** As task complexity increases from Level 0 to Level 3, we observe a consistent structural trend: functional correctness tends to degrade, while optimization and efficiency metrics become more discriminative. As shown in Table 12 and Table 25, smaller or non-reasoning models often see their performance collapse to near-zero at Level 3. Conversely, reasoning-enhanced models maintain robustness at higher complexities; for instance, Table 14 shows that Grok-4-1-Fast-Reasoning retains a 41.48% `eff@10` score even at Level 3. Higher complexity introduces deeper control branching and stronger cross-cycle dependencies, significantly raising the difficulty of validation. However, as evidenced by the high optimization ceilings in Table 21, complex designs also expose more *actionable optimization levers*, such as pipeline repartitioning, meaning that when models *do* succeed, the achievable gains are substantial.

**Impact of model scale.** We find that larger models are generally *beneficial* rather than irrelevant. They improve stability and raise the *correctness floor*, often delivering higher optimization scores once correctness is satisfied. This effect is clearly reflected in same-family comparisons. For example, comparing Table 16 (Llama-3.1-8B) with Table 18 (Llama-3.1-405B) reveals that the larger model attains substantially higher `opt@10` scores across all complexity levels (*e.g.*, improving Level 2 `opt@10` from 26.03% to 39.39%). Nevertheless, scale alone does not guarantee optimization; performance is strongly gated by the model's ability to maintain correctness under structural constraints.

**Per-category analysis.** Our results suggest an intuitive "optimisability" distinction rooted in RTL structure. Compute-dominated tasks (*e.g.*, arithmetic, dsp) feature regular datapaths and repeated operators, making them amenable to local rewrites. Consequently, models often yield stronger optimization results in these categories, as seen in the high arithmetic scores in Table 15. In contrast, control-dominated tasks (*e.g.*, processor) contain rigid, immovable structures that compress the valid rewrite space. The largest performance gap appears in this control path optimization.

## L Detailed design complexity criteria

We assign each RTL design a discrete complexity score $C \in \{0, 1, 2, 3, 4, 5\}$ to stratify evaluation difficulty. This score primarily reflects *structural RTL complexity*, and is not merely based on lines of code (LoC) or area. To determine this score, we rigorously evaluate four key dimensions: (i) control complexity (finite state machine (FSM) size/hierarchy, nested guards, timers/counters); (ii) datapath complexity (pipeline depth, dependencies/feedback, address generation/format conversion); (iii) protocol/algorithmic complexity (non-trivial parsing or algorithm kernels); and (iv) system integration (multi-module coordination, rich interfaces).

- **C=0 (Trivial Combinational).** Pure combinational logic with simple operations (*e.g.*, basic arithmetic), possessing no state or complex control structures.
- **C=1 (Simple Sequential).** Shallow sequential structure, typically characterized by a small FSM ($< 8$ states) and/or a 2–3 stage pipeline with limited nesting and weak cross-cycle dependencies.
- **C=2 (Moderate).** Designs requiring multiple non-trivial features, such as larger FSMs with nested transitions, pipelines with $\geq 4$ stages and data dependencies/feedback, or moderate protocol parsing/segmentation. These designs must include nested conditionals or non-trivial addressing/indexing/format conversion (*e.g.*, a simple toy CPU).
- **C=3 (High).** Complex designs featuring hierarchical FSMs, deeply nested control/dataflow, and non-trivial protocal/algorithmic processing (*e.g.*, LTSSM-like sequencing, DSP, image/video kernels). These typically involve timer/counter-driven control, timer/counter comparisons, multiple interacting datapaths, and substantial RTL structure (many `always` blocks, rich hierarchy).
- **C=4 (Repo-Level Subsystem).** Repository-scale subsystems integrating multiple complex submodules with rich interfaces/flow control. Examples include professional CPUs with advanced features (*e.g.*, sophisticated pipeline control plus branch prediction and/or caches), tiny GPUs, simple AI accelerators, or full communication subsystems.
- **C=5 (Industrial-Grade).** Highly integrated, near production-class open-source designs at the upper complexity ceiling, such as BlackParrot-class CPUs, MIAOW-like GPUs, industrial Ethernet subsystems or full PCIe controllers.

## M Evaluation with commercial synthesis tools

While the primary evaluation of TurboRTL utilizes the open-source toolchain to ensure maximum accessibility and reproducibility for the research community, we validate the benchmark's industrial fidelity using Synopsys Design Compiler (DC). This cross-toolchain analysis quantifies the impact of the synthesis engine on reported optimization gains and verifies that the performance hierarchy observed in open-source flows remains consistent under industry-standard environments. For this validation, we select a representative subset of the model suite—including top-performing candidates from the GPT, Grok, and DeepSeek families—to observe how the state-of-the-art performance ceiling shifts across different synthesis heuristics. Our reporting is restricted to optimization (`opt@k`) and

Table 7: Commercial synthesis tool evaluation results. Each cell reports the **DC Result** followed by the **Delta** ($\Delta$ = DC − Yosys) in parentheses. All values are reported in percentages.

| Model | opt@1 ($\Delta$) | opt@5 ($\Delta$) | opt@10 ($\Delta$) | eff@1 ($\Delta$) | eff@5 ($\Delta$) | eff@10 ($\Delta$) |
|---|---|---|---|---|---|---|
| GPT-5.2 | 24.29 (−1.09) | 46.81 (−3.85) | 56.42 (−3.36) | 14.49 (−5.51) | 30.88 (−11.29) | 39.91 (−11.80) |
| GPT-4o | 12.80 (±0.00) | 23.73 (−5.31) | 28.43 (−8.29) | 9.28 (−0.79) | 16.11 (−8.00) | 19.63 (−11.14) |
| GPT-4o-mini | 9.18 (+0.78) | 20.48 (+0.59) | 25.90 (+0.05) | 6.92 (+0.94) | 16.77 (+0.58) | 21.54 (−0.43) |
| Grok-4-1-Fast | 13.85 (+4.53) | 29.50 (+7.74) | 34.37 (+6.89) | 9.89 (+1.94) | 23.01 (+4.02) | 28.28 (+3.80) |
| Grok-4-1-Fast-Reasoning | 40.24 (+2.22) | 67.60 (−0.85) | 74.64 (−3.32) | 27.52 (−4.36) | 49.62 (−11.36) | 56.96 (−15.20) |
| Grok-Code-Fast-1 | 23.07 (−3.28) | 53.26 (−5.20) | 67.67 (−3.24) | 14.85 (−6.58) | 37.02 (−12.67) | 48.95 (−13.34) |
| DeepSeek-Chat | 14.90 (−0.79) | 29.29 (−3.35) | 36.20 (−4.14) | 10.75 (−2.15) | 24.40 (−3.46) | 31.86 (−3.56) |
| DeepSeek-Reasoner | 26.25 (−1.37) | 54.39 (−0.54) | 64.94 (−0.68) | 17.35 (−5.34) | 41.03 (−6.54) | 51.66 (−6.85) |

efficiency (`eff@k`) metrics, as synthesizability (`syn@k`) and functional correctness (`fun@k`) exhibit near-perfect agreement across both Yosys and DC. This consistency is expected, as these metrics assess the syntactic and logical validity of the generated Verilog, which is fundamentally independent of the underlying synthesis algorithms.

The empirical results, detailed in Table 7, reveal several critical insights into the relationship between LLM-driven optimization and commercial synthesis engines. For the majority of models, we observe slightly negative deltas ($\Delta$ = DC − Yosys), which we attribute to "headroom compression". Commercial compilers like Design Compiler typically apply more aggressive baseline optimizations to the suboptimal input RTL than open-source counterparts. By reducing the "low-hanging fruit" available for optimization, these tools naturally narrow the relative margin for improvement that an LLM can claim. However, despite these shifts in absolute percentages, the relative performance hierarchy remains remarkably stable. Models that dominate in the Yosys environment consistently maintain their lead in the DC environment, confirming that TurboRTL captures fundamental architectural inefficiencies rather than tool-specific artifacts. Furthermore, the fact that state-of-the-art models continue to achieve significant gains—such as GPT-5.2 reaching a 24.29% `opt@1` success rate even under DC—underscores that the EqSat-driven challenges in the benchmark possess a structural complexity that remains non-trivial for modern industry compilers.

## N Comparison with specification-only generation

To isolate a model's optimization ability from its general generation ability, we conduct a "Spec-to-RTL" control experiment using GPT-4o, chosen for its satisfactory performance and faster inference speed. Specifically, we first utilize Gemini-3-pro to generate a detailed natural language specification (spec) for every design in TurboRTL, including its functional behavior, input/output ports, *etc.* These specs are manually verified and refined for accuracy. GPT-4o is subsequently tasked with generating RTL code solely from these specifications without access to the original suboptimal baseline.

As shown in Table 9, while "Spec-to-RTL" achieves high synthesizability, its success rate in reaching the optimization target is significantly lower, which indicates that generating a correct and optimized implementation from scratch (specification) is substantially more difficult than refining an existing one. This demonstrates that models rely heavily on the structural semantics provided by the baseline RTL. Refining an existing design to be more efficient is a distinct and more achievable task than attempting to generate an optimal implementation solely from functional requirements. These results confirm that TurboRTL effectively benchmarks a model's ability to analyze and refactor existing logic, rather than simply recalling "known good" implementations from its training data.

## O Query budget versus token budget

The query budget used in the main evaluation controls the number of model calls: each method receives at most 20 queries per task. This design provides an equal-query comparison, but it does not guarantee equal token consumption across providers or interaction paradigms. To examine whether our conclusions change under a token-normalized budget, we conduct an additional experiment on the same frozen subset of 51 complexity-$C = 2$ tasks. We compare the single-shot, population-based, and experience-driven agentic settings under either the original 20-call budget or a 100K-token budget.

Table 8: Comparison under equal-query and token-normalized budgets on the same frozen 51-task complexity-$C = 2$ subset. Each entry reports `opt`/`eff` (%).

| Model | Method | 20 calls | 100K tokens |
|---|---|---|---|
| GPT-5.5 | Single | 62.75 / 55.47 | 58.82 / 51.55 |
| | Population | 60.78 / 57.48 | 60.78 / 56.26 |
| | Agentic | 78.43 / 75.55 | 76.47 / 70.70 |
| Claude Opus 4.8 | Single | 54.90 / 52.32 | 54.90 / 51.08 |
| | Population | 64.71 / 63.22 | 50.98 / 47.75 |
| | Agentic | 68.63 / 67.39 | 64.71 / 63.47 |

Table 9: Comparison between RTL Optimization and Spec-to-RTL Generation (GPT-4o). All values are reported in percentages.

| Model | syn@1 | syn@5 | syn@10 | fun@1 | fun@5 | fun@10 | opt@1 | opt@5 | opt@10 | eff@1 | eff@5 | eff@10 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|
| Optimization | 86.08 | 98.16 | 99.31 | 23.94 | 42.89 | 50.37 | 12.80 | 29.04 | 36.72 | 10.07 | 24.11 | 30.77 |
| Spec-to-RTL | 88.25 | 97.96 | 98.82 | 6.58 | 9.29 | 10.60 | 2.47 | 4.71 | 6.04 | 2.14 | 4.23 | 5.48 |

For the 100K-token condition, we include complete calls for each task in sequence until the next call would raise the cumulative gateway-reported token count above 100K. The results are reported as `opt`/`eff` percentages in Table 8. The agentic workflow remains the strongest setting for both models under both budgets. Its performance declines only modestly under the token cap, while the other methods show no consistent advantage. These results suggest that the observed agentic gains are not simply attributable to higher reported-token consumption.

## P Relative performance computation with eff

The `eff` metric should be interpreted as a relative performance measure rather than an absolute measure of optimization quality. For a given original design $d$, the true maximum achievable optimization is generally unknown: the space of functionally equivalent RTL implementations is open-ended, and no objective upper bound on the attainable ADP improvement is available. It is therefore necessary to define a validated reference point in order to place the improvements found by different models on a common scale.

More generally, let $d_{\text{ref}}$ denote any synthesizable and functionally equivalent reference implementation of $d$ with lower ADP. The reference may be obtained from an e-graph optimizer, a human hardware engineer, another model, or a different EDA tool. We define the relative efficiency of a candidate $s_i$ with respect to this reference as

$$e_i^{(d_{\text{ref}})} = \begin{cases} \min\left\{1, \dfrac{\text{ADP}(d) - \text{ADP}(s_i)}{\text{ADP}(d) - \text{ADP}(d_{\text{ref}})}\right\} & \text{if } \Phi_{\text{opt}}(s_i) \text{ is True,} \\ 0 & \text{otherwise.} \end{cases} \tag{7}$$

The corresponding `eff@k` is obtained by substituting $e_i^{(d_{\text{ref}})}$ into Eq. 4. Thus, a score of 1 means that the candidate matches or surpasses the chosen reference, whereas intermediate values quantify the fraction of the reference improvement gap that has been closed. The clipping at 1 reflects the intended interpretation that surpassing the reference receives full credit rather than producing an unbounded score.

Throughout this paper, we instantiate $d_{\text{ref}}$ as $d_{\text{egg}}$, the verified lower-ADP implementation produced by the EqSat-driven filtering pipeline. This choice provides a reproducible and automatically generated target, but it does not imply that $d_{\text{egg}}$ is a global optimum or an upper bound. The e-graph reference is generated by a limited rule set and cost model, and a model may discover a structurally different implementation with an even lower ADP. Accordingly, `eff` measures the fraction of a known, validated optimization gap closed by a model, while `opt` independently records whether the model improves upon the original design. Reporting both metrics avoids conflating reference-relative progress with the binary question of whether any improvement was achieved.

Table 10: Detailed performance results for `GPT-5.2`. All values in %. Dark gray = Best, Light gray = Second Best.

| Split | syn@1 | syn@5 | syn@10 | fun@1 | fun@5 | fun@10 | opt@1 | opt@5 | opt@10 | eff@1 | eff@5 | eff@10 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|
| **Complexity** | | | | | | | | | | | | |
| 0 | 93.71 | 99.54 | 99.97 | 47.13 | 69.41 | 76.41 | 21.52 | 42.89 | 50.84 | 18.64 | 38.33 | 45.86 |
| 1 | 89.61 | 99.27 | 99.82 | 48.20 | 75.05 | 82.18 | 29.23 | 55.27 | 64.60 | 23.24 | 46.72 | 57.04 |
| 2 | 83.63 | 97.54 | 98.03 | 45.98 | 67.20 | 71.82 | 22.65 | 52.84 | 63.13 | 14.01 | 36.74 | 47.52 |
| 3 | 85.83 | 100.00 | 100.00 | 27.50 | 58.64 | 64.91 | 15.00 | 38.30 | 49.73 | 14.74 | 37.32 | 48.05 |
| **Category** | | | | | | | | | | | | |
| arithmetic | 94.48 | 99.58 | 99.98 | 42.85 | 68.41 | 76.07 | 19.77 | 39.34 | 46.64 | 17.20 | 35.64 | 42.45 |
| dsp | 89.70 | 99.97 | 100.00 | 53.80 | 85.12 | 90.84 | 33.40 | 66.10 | 76.58 | 27.40 | 58.69 | 71.71 |
| communication | 89.79 | 99.99 | 100.00 | 53.85 | 73.02 | 78.19 | 25.63 | 54.93 | 64.71 | 17.27 | 39.84 | 51.17 |
| miscellaneous | 90.92 | 99.54 | 99.99 | 54.61 | 74.95 | 82.18 | 36.05 | 62.37 | 72.31 | 27.85 | 50.99 | 61.01 |
| datapath | 81.40 | 94.94 | 95.96 | 40.00 | 66.09 | 73.42 | 23.80 | 46.70 | 55.25 | 17.77 | 37.14 | 46.32 |
| memory | 86.47 | 96.61 | 98.61 | 25.88 | 48.05 | 56.55 | 13.82 | 39.19 | 52.12 | 12.68 | 34.41 | 44.11 |
| control | 83.00 | 98.31 | 99.89 | 39.67 | 53.83 | 58.12 | 21.67 | 37.83 | 42.76 | 18.15 | 32.65 | 37.71 |
| processor | 72.00 | 98.90 | 99.99 | 65.00 | 98.87 | 99.99 | 13.00 | 33.62 | 45.26 | 5.59 | 20.35 | 33.93 |
| crypto | 90.00 | 100.00 | 100.00 | 38.75 | 87.62 | 98.51 | 30.00 | 81.55 | 96.83 | 16.35 | 47.32 | 59.76 |

Table 11: Detailed performance results for `GPT-4o`. All values in %. Dark gray = Best, Light gray = Second Best.

| Split | syn@1 | syn@5 | syn@10 | fun@1 | fun@5 | fun@10 | opt@1 | opt@5 | opt@10 | eff@1 | eff@5 | eff@10 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|
| **Complexity** | | | | | | | | | | | | |
| 0 | 92.58 | 99.37 | 99.88 | 32.30 | 48.33 | 53.30 | 13.37 | 26.68 | 32.94 | 11.12 | 22.54 | 28.01 |
| 1 | 86.16 | 98.89 | 99.81 | 20.60 | 41.30 | 49.97 | 12.39 | 29.51 | 37.84 | 9.46 | 24.09 | 30.94 |
| 2 | 78.63 | 95.47 | 97.40 | 21.08 | 41.17 | 49.53 | 14.02 | 33.62 | 41.80 | 10.71 | 28.09 | 36.00 |
| 3 | 50.83 | 85.76 | 95.33 | 3.33 | 14.18 | 23.25 | 3.33 | 14.18 | 23.25 | 3.33 | 14.18 | 23.25 |
| **Category** | | | | | | | | | | | | |
| arithmetic | 93.20 | 99.67 | 99.99 | 26.22 | 40.30 | 45.65 | 12.21 | 23.98 | 30.10 | 9.96 | 19.56 | 24.41 |
| dsp | 79.30 | 97.86 | 99.43 | 17.00 | 37.70 | 48.71 | 8.60 | 23.62 | 33.21 | 7.58 | 19.96 | 27.28 |
| communication | 81.15 | 98.11 | 99.68 | 31.87 | 56.06 | 62.65 | 20.31 | 47.59 | 56.18 | 14.55 | 38.90 | 48.34 |
| miscellaneous | 91.45 | 98.94 | 99.72 | 28.55 | 52.32 | 60.91 | 16.32 | 38.58 | 49.68 | 13.62 | 33.55 | 42.49 |
| datapath | 83.60 | 96.62 | 98.88 | 24.20 | 41.13 | 50.22 | 13.40 | 26.95 | 33.32 | 11.04 | 23.97 | 29.96 |
| memory | 74.41 | 89.81 | 93.24 | 6.18 | 20.95 | 27.38 | 5.00 | 17.42 | 22.67 | 4.57 | 16.32 | 21.96 |
| control | 86.33 | 99.60 | 100.00 | 16.33 | 36.61 | 43.21 | 7.00 | 15.37 | 19.89 | 5.01 | 12.04 | 15.77 |
| processor | 82.00 | 99.97 | 100.00 | 27.00 | 48.85 | 55.26 | 2.00 | 8.95 | 15.26 | 2.00 | 8.95 | 15.26 |
| crypto | 95.00 | 100.00 | 100.00 | 21.25 | 36.18 | 44.08 | 18.75 | 31.24 | 37.50 | 2.35 | 4.93 | 7.15 |

Table 12: Detailed performance results for `GPT-4o-mini`. All values in %. Dark gray = Best, Light gray = Second Best.

| Split | syn@1 | syn@5 | syn@10 | fun@1 | fun@5 | fun@10 | opt@1 | opt@5 | opt@10 | eff@1 | eff@5 | eff@10 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|
| **Complexity** | | | | | | | | | | | | |
| 0 | 54.83 | 59.26 | 59.53 | 19.94 | 35.09 | 40.73 | 8.93 | 19.09 | 23.43 | 7.22 | 15.94 | 20.17 |
| 1 | 72.11 | 85.70 | 86.93 | 15.00 | 31.77 | 38.93 | 6.87 | 19.00 | 26.68 | 4.52 | 14.48 | 21.13 |
| 2 | 60.88 | 83.93 | 88.69 | 16.86 | 29.69 | 34.60 | 12.75 | 26.11 | 30.80 | 8.59 | 23.27 | 30.04 |
| 3 | 32.50 | 70.31 | 79.27 | 0.00 | 0.00 | 0.00 | 0.00 | 0.00 | 0.00 | 0.00 | 0.00 | 0.00 |
| **Category** | | | | | | | | | | | | |
| arithmetic | 55.93 | 60.14 | 60.44 | 11.98 | 22.87 | 27.92 | 3.90 | 12.23 | 17.57 | 3.00 | 9.40 | 13.71 |
| dsp | 63.20 | 84.44 | 88.26 | 10.90 | 28.40 | 36.88 | 2.60 | 10.64 | 17.41 | 2.18 | 8.84 | 14.48 |
| communication | 68.02 | 91.67 | 94.43 | 24.38 | 42.99 | 48.53 | 17.19 | 34.78 | 40.61 | 11.60 | 30.40 | 38.79 |
| miscellaneous | 77.76 | 86.45 | 86.84 | 20.13 | 43.84 | 54.07 | 11.84 | 29.71 | 39.34 | 8.05 | 24.00 | 33.92 |
| datapath | 64.80 | 74.35 | 75.80 | 21.40 | 39.07 | 46.03 | 15.80 | 29.17 | 34.98 | 13.39 | 26.26 | 31.76 |
| memory | 50.88 | 71.27 | 76.63 | 5.59 | 12.70 | 14.70 | 5.59 | 12.70 | 14.70 | 4.29 | 9.65 | 12.04 |
| control | 69.33 | 79.89 | 80.00 | 20.00 | 29.37 | 32.52 | 7.00 | 18.44 | 22.94 | 4.86 | 12.87 | 15.46 |
| processor | 71.00 | 79.99 | 80.00 | 31.00 | 39.84 | 40.00 | 1.00 | 5.00 | 10.00 | 0.58 | 2.91 | 5.81 |
| crypto | 87.50 | 99.59 | 100.00 | 42.50 | 50.00 | 50.00 | 20.00 | 25.00 | 25.00 | 2.26 | 3.13 | 3.54 |

Table 13: Detailed performance results for `Grok-4-1-Fast`. All values in %. Dark gray = Best, Light gray = Second Best.

| Split | syn@1 | syn@5 | syn@10 | fun@1 | fun@5 | fun@10 | opt@1 | opt@5 | opt@10 | eff@1 | eff@5 | eff@10 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|
| **Complexity** | | | | | | | | | | | | |
| 0 | 90.67 | 99.59 | 99.95 | 18.76 | 38.81 | 47.20 | 7.81 | 19.38 | 25.83 | 6.86 | 17.17 | 23.22 |
| 1 | 85.25 | 99.13 | 99.84 | 14.01 | 29.67 | 36.69 | 10.32 | 22.96 | 29.29 | 9.04 | 20.04 | 25.53 |
| 2 | 71.76 | 97.73 | 99.69 | 13.73 | 28.73 | 31.86 | 10.29 | 25.15 | 28.56 | 7.75 | 21.49 | 26.62 |
| 3 | 31.67 | 71.29 | 81.80 | 1.67 | 8.33 | 16.67 | 0.00 | 0.00 | 0.00 | 0.00 | 0.00 | 0.00 |
| **Category** | | | | | | | | | | | | |
| arithmetic | 89.77 | 99.86 | 100 | 13.26 | 30.33 | 38.42 | 5.47 | 15.05 | 21.15 | 4.75 | 12.86 | 18.41 |
| dsp | 71.50 | 95.07 | 97.53 | 12.80 | 27.12 | 34.63 | 8.00 | 19.32 | 24.42 | 7.42 | 17.35 | 21.76 |
| communication | 78.85 | 97.82 | 99.66 | 24.48 | 43.40 | 46.50 | 17.71 | 35.61 | 39.62 | 14.59 | 32.17 | 38.02 |
| miscellaneous | 92.24 | 99.95 | 100 | 15.26 | 34.26 | 40.97 | 13.16 | 29.72 | 37.35 | 12.54 | 28.78 | 36.12 |
| datapath | 85.00 | 99.52 | 99.99 | 16.80 | 33.33 | 40.73 | 10.80 | 22.38 | 28.11 | 9.54 | 20.64 | 26.04 |
| memory | 77.94 | 96.72 | 99.35 | 7.35 | 21.11 | 26.52 | 4.71 | 15.07 | 20.49 | 2.96 | 11.65 | 18.37 |
| control | 83.00 | 99.08 | 99.96 | 4.67 | 14.45 | 21.75 | 4.33 | 12.78 | 18.42 | 2.57 | 8.41 | 12.86 |
| processor | 84.00 | 99.67 | 100 | 33.00 | 53.92 | 65.26 | 1.00 | 5.00 | 10.00 | 0.04 | 0.19 | 0.37 |
| crypto | 81.25 | 99.96 | 100 | 13.75 | 38.74 | 47.36 | 11.25 | 34.11 | 44.04 | 1.34 | 4.26 | 5.75 |

Table 14: Detailed performance results for `Grok-4-1-Fast-Reasoning`. All values in %. Dark gray = Best, Light gray = Second Best.

| Split | syn@1 | syn@5 | syn@10 | fun@1 | fun@5 | fun@10 | opt@1 | opt@5 | opt@10 | eff@1 | eff@5 | eff@10 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|
| **Complexity** | | | | | | | | | | | | |
| 0 | 94.83 | 99.45 | 99.88 | 73.37 | 89.13 | 91.51 | 36.46 | 65.21 | 75.36 | 32.12 | 60.83 | 71.67 |
| 1 | 87.85 | 98.02 | 98.85 | 64.19 | 88.31 | 93.11 | 42.46 | 72.66 | 81.72 | 36.22 | 65.09 | 75.27 |
| 2 | 73.82 | 96.62 | 98.75 | 49.41 | 77.23 | 83.39 | 32.06 | 67.20 | 76.19 | 22.41 | 53.91 | 67.95 |
| 3 | 37.50 | 64.29 | 66.64 | 27.50 | 53.94 | 64.16 | 6.67 | 27.49 | 42.54 | 6.26 | 26.24 | 41.48 |
| **Category** | | | | | | | | | | | | |
| arithmetic | 91.05 | 98.39 | 99.29 | 71.69 | 89.72 | 92.53 | 31.98 | 61.11 | 72.50 | 29.08 | 56.95 | 68.65 |
| dsp | 81.10 | 95.52 | 97.51 | 67.40 | 91.18 | 95.09 | 44.80 | 75.45 | 83.47 | 38.66 | 65.92 | 74.56 |
| communication | 80.94 | 95.68 | 96.84 | 57.50 | 78.62 | 83.84 | 39.27 | 68.94 | 74.96 | 28.14 | 56.04 | 67.83 |
| miscellaneous | 92.37 | 99.54 | 99.99 | 64.47 | 90.44 | 96.27 | 47.24 | 80.26 | 90.67 | 40.63 | 72.07 | 83.55 |
| datapath | 92.60 | 99.86 | 100.00 | 59.20 | 80.73 | 84.96 | 39.40 | 66.16 | 74.26 | 31.99 | 59.24 | 67.80 |
| memory | 76.76 | 93.30 | 94.11 | 51.47 | 83.33 | 87.36 | 30.88 | 69.81 | 81.47 | 25.53 | 65.69 | 79.76 |
| control | 76.00 | 99.19 | 100.00 | 38.67 | 72.60 | 83.32 | 30.00 | 60.47 | 71.60 | 28.78 | 58.82 | 70.54 |
| processor | 99.00 | 100.00 | 100.00 | 91.00 | 99.99 | 100.00 | 29.00 | 70.71 | 87.03 | 20.27 | 60.34 | 80.52 |
| crypto | 92.50 | 100.00 | 100.00 | 48.75 | 56.25 | 62.50 | 43.75 | 56.25 | 62.50 | 21.67 | 39.84 | 51.79 |

Table 15: Detailed performance results for `Grok-Code-Fast-1`. All values in %. Dark gray = Best, Light gray = Second Best.

| Split | syn@1 | syn@5 | syn@10 | fun@1 | fun@5 | fun@10 | opt@1 | opt@5 | opt@10 | eff@1 | eff@5 | eff@10 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|
| **Complexity** | | | | | | | | | | | | |
| 0 | 93.88 | 99.88 | 100.00 | 50.67 | 81.08 | 88.86 | 20.90 | 47.19 | 58.99 | 17.87 | 41.18 | 52.36 |
| 1 | 91.41 | 99.55 | 99.92 | 53.20 | 85.30 | 92.50 | 31.94 | 65.80 | 78.02 | 26.40 | 57.17 | 70.00 |
| 2 | 88.24 | 99.60 | 99.99 | 50.69 | 80.77 | 86.25 | 21.27 | 58.58 | 72.99 | 14.29 | 43.78 | 58.27 |
| 3 | 64.17 | 83.32 | 83.33 | 38.33 | 70.02 | 75.00 | 18.33 | 51.06 | 61.98 | 17.26 | 49.23 | 61.10 |
| **Category** | | | | | | | | | | | | |
| arithmetic | 91.74 | 99.24 | 99.87 | 48.31 | 78.22 | 87.43 | 20.47 | 43.41 | 54.69 | 17.80 | 38.51 | 49.17 |
| dsp | 87.50 | 97.99 | 98.00 | 53.20 | 91.77 | 97.03 | 31.90 | 71.39 | 83.91 | 31.90 | 71.39 | 83.91 |
| communication | 92.71 | 99.82 | 100.00 | 59.48 | 87.79 | 94.49 | 28.96 | 68.88 | 82.97 | 21.49 | 52.91 | 67.66 |
| miscellaneous | 94.87 | 99.96 | 100.00 | 56.97 | 87.48 | 94.61 | 35.26 | 70.03 | 83.22 | 29.19 | 62.16 | 76.70 |
| datapath | 87.00 | 99.47 | 99.98 | 46.20 | 76.42 | 82.81 | 23.80 | 54.91 | 67.75 | 18.73 | 46.22 | 59.53 |
| memory | 88.24 | 99.70 | 100.00 | 36.47 | 63.09 | 71.97 | 15.29 | 42.74 | 56.20 | 13.10 | 38.02 | 50.33 |
| control | 91.67 | 99.99 | 100.00 | 45.33 | 77.32 | 79.88 | 31.00 | 66.75 | 76.03 | 26.80 | 60.95 | 70.14 |
| processor | 93.00 | 100.00 | 100.00 | 80.00 | 99.96 | 100.00 | 6.00 | 24.37 | 39.13 | 2.30 | 10.16 | 17.98 |
| crypto | 96.25 | 100.00 | 100.00 | 47.50 | 92.62 | 99.79 | 38.75 | 86.04 | 98.47 | 15.16 | 47.47 | 60.59 |

Table 16: Detailed performance results for `Llama-3.1-8B`. All values in %. Dark gray = Best, Light gray = Second Best.

| Split | syn@1 | syn@5 | syn@10 | fun@1 | fun@5 | fun@10 | opt@1 | opt@5 | opt@10 | eff@1 | eff@5 | eff@10 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|
| **Complexity** | | | | | | | | | | | | |
| 0 | 72.36 | 96.40 | 98.96 | 20.39 | 57.03 | 72.80 | 5.90 | 21.55 | 32.84 | 4.69 | 17.50 | 27.06 |
| 1 | 61.23 | 94.96 | 98.66 | 13.91 | 45.22 | 63.08 | 4.40 | 17.65 | 28.74 | 3.32 | 13.50 | 22.16 |
| 2 | 50.59 | 87.74 | 95.24 | 14.71 | 48.62 | 66.43 | 4.02 | 16.28 | 26.03 | 3.02 | 12.31 | 19.67 |
| 3 | 37.50 | 82.10 | 95.06 | 15.00 | 42.55 | 55.86 | 5.83 | 18.68 | 24.91 | 5.62 | 17.63 | 22.80 |
| **Category** | | | | | | | | | | | | |
| arithmetic | 71.28 | 96.72 | 99.07 | 17.21 | 52.71 | 70.30 | 4.42 | 17.39 | 27.88 | 3.49 | 13.71 | 21.99 |
| dsp | 52.90 | 94.49 | 99.47 | 12.50 | 39.69 | 55.59 | 2.80 | 11.19 | 18.77 | 2.30 | 9.23 | 15.45 |
| communication | 41.25 | 81.16 | 92.51 | 12.92 | 44.69 | 61.17 | 4.79 | 20.35 | 33.81 | 3.42 | 14.88 | 25.41 |
| miscellaneous | 74.47 | 99.76 | 100.00 | 16.58 | 53.83 | 73.63 | 6.05 | 23.76 | 37.00 | 4.60 | 17.78 | 27.03 |
| datapath | 65.80 | 95.44 | 98.81 | 19.40 | 52.56 | 70.57 | 7.00 | 22.00 | 30.79 | 5.71 | 18.70 | 26.62 |
| memory | 60.88 | 91.15 | 96.61 | 15.29 | 46.35 | 59.32 | 5.00 | 18.46 | 27.11 | 3.64 | 14.22 | 21.67 |
| control | 68.67 | 98.08 | 99.89 | 27.67 | 68.81 | 86.40 | 9.00 | 33.27 | 50.64 | 7.43 | 29.11 | 46.05 |
| processor | 68.00 | 99.08 | 100.00 | 14.00 | 48.25 | 68.39 | 2.00 | 8.95 | 15.26 | 1.20 | 5.76 | 10.95 |
| crypto | 76.25 | 99.79 | 100.00 | 11.25 | 36.79 | 47.23 | 1.25 | 6.25 | 12.50 | 0.14 | 0.68 | 1.36 |

Table 17: Detailed performance results for `Llama-3.1-70B`. All values in %. Dark gray = Best, Light gray = Second Best.

| Split | syn@1 | syn@5 | syn@10 | fun@1 | fun@5 | fun@10 | opt@1 | opt@5 | opt@10 | eff@1 | eff@5 | eff@10 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|
| **Complexity** | | | | | | | | | | | | |
| 0 | 89.49 | 99.70 | 100.00 | 28.37 | 62.09 | 74.15 | 11.01 | 30.77 | 43.48 | 8.91 | 26.90 | 39.05 |
| 1 | 81.58 | 98.94 | 99.57 | 19.65 | 49.36 | 60.25 | 11.30 | 32.37 | 42.46 | 8.75 | 25.68 | 34.48 |
| 2 | 72.16 | 95.33 | 98.11 | 20.20 | 48.21 | 59.87 | 13.24 | 34.55 | 44.25 | 9.73 | 28.11 | 38.07 |
| 3 | 65.83 | 93.28 | 98.25 | 24.17 | 53.96 | 62.45 | 10.83 | 34.16 | 48.24 | 9.75 | 28.94 | 37.85 |
| **Category** | | | | | | | | | | | | |
| arithmetic | 88.55 | 99.78 | 100.00 | 23.60 | 56.45 | 69.47 | 9.53 | 29.60 | 40.65 | 6.85 | 23.95 | 34.41 |
| dsp | 74.60 | 98.78 | 99.79 | 18.90 | 43.27 | 52.05 | 8.50 | 23.05 | 32.47 | 7.70 | 20.18 | 27.24 |
| communication | 78.02 | 99.34 | 99.96 | 28.85 | 61.26 | 71.27 | 18.85 | 47.12 | 58.39 | 13.34 | 36.79 | 47.85 |
| miscellaneous | 90.00 | 99.95 | 100.00 | 20.92 | 53.88 | 66.40 | 14.47 | 37.76 | 48.79 | 12.57 | 32.57 | 42.73 |
| datapath | 78.40 | 96.56 | 99.16 | 24.20 | 52.45 | 61.13 | 11.00 | 29.15 | 39.73 | 10.19 | 27.44 | 38.03 |
| memory | 63.82 | 85.04 | 92.10 | 16.76 | 40.15 | 46.68 | 6.76 | 23.04 | 33.59 | 4.09 | 16.55 | 27.01 |
| control | 88.00 | 100.00 | 100.00 | 20.33 | 57.35 | 73.37 | 10.00 | 31.53 | 42.93 | 8.92 | 29.07 | 40.13 |
| processor | 83.00 | 99.93 | 100.00 | 15.00 | 53.22 | 72.51 | 8.00 | 31.38 | 47.03 | 4.59 | 19.06 | 30.42 |
| crypto | 83.75 | 99.96 | 100.00 | 16.25 | 47.94 | 69.08 | 11.25 | 39.21 | 56.44 | 4.46 | 19.37 | 33.70 |

Table 18: Detailed performance results for `Llama-3.1-405B`. All values in %. Dark gray = Best, Light gray = Second Best.

| Split | syn@1 | syn@5 | syn@10 | fun@1 | fun@5 | fun@10 | opt@1 | opt@5 | opt@10 | eff@1 | eff@5 | eff@10 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|
| **Complexity** | | | | | | | | | | | | |
| 0 | 87.30 | 98.38 | 98.85 | 30.11 | 54.75 | 64.43 | 12.64 | 29.71 | 38.16 | 10.12 | 25.74 | 34.38 |
| 1 | 83.73 | 98.16 | 99.38 | 21.62 | 45.10 | 55.61 | 14.05 | 30.38 | 38.57 | 12.12 | 26.09 | 33.03 |
| 2 | 78.04 | 97.76 | 98.98 | 18.04 | 37.97 | 46.31 | 13.73 | 30.99 | 39.39 | 10.21 | 26.57 | 35.70 |
| 3 | 53.33 | 89.81 | 98.13 | 8.33 | 24.15 | 33.33 | 6.67 | 22.85 | 33.24 | 6.67 | 22.85 | 33.24 |
| **Category** | | | | | | | | | | | | |
| arithmetic | 87.97 | 99.60 | 99.99 | 23.90 | 45.79 | 56.50 | 9.48 | 23.10 | 30.16 | 7.59 | 19.61 | 26.46 |
| dsp | 76.30 | 94.19 | 96.61 | 15.60 | 31.26 | 41.21 | 10.10 | 21.47 | 29.01 | 9.39 | 18.93 | 24.55 |
| communication | 78.65 | 98.15 | 99.47 | 27.81 | 56.05 | 65.99 | 20.42 | 42.64 | 51.07 | 15.32 | 35.29 | 43.73 |
| miscellaneous | 89.21 | 99.96 | 100.00 | 32.63 | 64.03 | 75.39 | 22.24 | 46.76 | 58.20 | 19.06 | 42.69 | 54.76 |
| datapath | 84.20 | 99.06 | 99.93 | 31.40 | 52.43 | 58.45 | 19.00 | 38.51 | 45.57 | 16.64 | 35.91 | 44.40 |
| memory | 63.82 | 91.67 | 96.75 | 10.59 | 23.38 | 26.41 | 6.76 | 18.46 | 23.27 | 5.41 | 13.80 | 16.36 |
| control | 92.33 | 100.00 | 100.00 | 14.00 | 43.73 | 56.99 | 5.00 | 20.35 | 32.70 | 4.01 | 17.22 | 29.05 |
| processor | 93.00 | 100.00 | 100.00 | 17.00 | 45.80 | 59.67 | 4.00 | 18.95 | 35.26 | 4.00 | 18.95 | 35.26 |
| crypto | 93.75 | 100.00 | 100.00 | 12.50 | 34.91 | 44.07 | 7.50 | 29.14 | 43.00 | 3.05 | 13.14 | 21.69 |

Table 19: Detailed performance results for `Llama-4-maverick`. All values in %. Dark gray = Best, Light gray = Second Best.

| Split | syn@1 | syn@5 | syn@10 | fun@1 | fun@5 | fun@10 | opt@1 | opt@5 | opt@10 | eff@1 | eff@5 | eff@10 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|
| **Complexity** | | | | | | | | | | | | |
| 0 | 91.07 | 98.19 | 98.81 | 31.29 | 49.39 | 55.45 | 11.97 | 25.18 | 33.04 | 9.52 | 21.26 | 28.33 |
| 1 | 87.54 | 98.64 | 99.12 | 23.52 | 47.62 | 56.94 | 11.73 | 31.55 | 43.19 | 8.54 | 24.52 | 34.31 |
| 2 | 81.57 | 96.30 | 97.06 | 21.86 | 38.17 | 43.20 | 14.12 | 31.28 | 36.66 | 11.53 | 27.72 | 33.80 |
| 3 | 50.00 | 74.71 | 83.33 | 0.83 | 4.17 | 8.33 | 0.83 | 4.17 | 8.35 | 0.83 | 4.17 | 8.33 |
| **Category** | | | | | | | | | | | | |
| arithmetic | 91.51 | 98.53 | 98.82 | 25.76 | 41.99 | 48.31 | 9.07 | 21.74 | 29.54 | 6.70 | 17.54 | 24.31 |
| dsp | 79.90 | 96.66 | 98.00 | 25.30 | 54.64 | 66.77 | 9.60 | 29.49 | 42.34 | 8.17 | 24.09 | 34.10 |
| communication | 84.69 | 97.25 | 98.46 | 34.27 | 55.13 | 58.52 | 19.17 | 42.67 | 51.97 | 14.07 | 34.77 | 43.40 |
| miscellaneous | 89.87 | 96.48 | 97.25 | 27.24 | 50.51 | 59.57 | 18.42 | 38.16 | 47.63 | 15.82 | 34.19 | 43.86 |
| datapath | 87.20 | 99.72 | 100.00 | 19.00 | 35.55 | 42.56 | 9.80 | 23.60 | 30.28 | 8.75 | 20.99 | 27.29 |
| memory | 80.29 | 93.25 | 94.09 | 9.12 | 19.34 | 23.49 | 7.35 | 17.79 | 23.18 | 3.59 | 10.76 | 15.41 |
| control | 88.00 | 99.12 | 99.96 | 12.33 | 33.12 | 44.45 | 5.33 | 17.08 | 25.26 | 3.99 | 14.16 | 21.97 |
| processor | 77.00 | 99.96 | 100.00 | 26.00 | 39.95 | 40.00 | 4.00 | 17.89 | 30.53 | 2.75 | 12.31 | 21.01 |
| crypto | 100.00 | 100.00 | 100.00 | 35.00 | 76.82 | 94.60 | 26.25 | 65.29 | 87.07 | 9.67 | 37.34 | 60.70 |

Table 20: Detailed performance results for `Deepseek-Chat`. All values in %. Dark gray = Best, Light gray = Second Best.

| Split | syn@1 | syn@5 | syn@10 | fun@1 | fun@5 | fun@10 | opt@1 | opt@5 | opt@10 | eff@1 | eff@5 | eff@10 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|
| **Complexity** | | | | | | | | | | | | |
| 0 | 95.73 | 99.81 | 100.00 | 32.64 | 53.31 | 59.23 | 13.82 | 30.95 | 38.13 | 11.96 | 27.06 | 34.29 |
| 1 | 92.11 | 99.51 | 99.83 | 24.86 | 44.32 | 52.85 | 18.27 | 35.02 | 43.02 | 15.52 | 30.50 | 37.96 |
| 2 | 90.49 | 98.50 | 99.71 | 21.76 | 40.87 | 47.36 | 13.14 | 30.76 | 38.22 | 8.33 | 23.40 | 31.44 |
| 3 | 78.33 | 99.50 | 100.00 | 4.17 | 17.47 | 27.63 | 4.17 | 17.47 | 27.63 | 3.51 | 15.45 | 25.90 |
| **Category** | | | | | | | | | | | | |
| arithmetic | 94.88 | 99.89 | 100.00 | 29.88 | 49.83 | 55.90 | 13.95 | 31.44 | 39.60 | 11.13 | 26.31 | 34.33 |
| dsp | 89.00 | 99.77 | 100.00 | 24.90 | 44.07 | 51.92 | 17.00 | 32.56 | 38.95 | 15.48 | 29.82 | 36.48 |
| communication | 93.85 | 99.98 | 100.00 | 32.08 | 49.56 | 54.92 | 24.38 | 45.25 | 51.63 | 18.22 | 37.86 | 46.31 |
| miscellaneous | 93.95 | 99.70 | 100.00 | 25.00 | 42.71 | 50.98 | 19.34 | 33.60 | 41.00 | 17.74 | 31.66 | 38.01 |
| datapath | 90.60 | 97.28 | 99.41 | 22.40 | 47.24 | 61.45 | 11.40 | 28.73 | 39.44 | 9.68 | 25.29 | 35.74 |
| memory | 85.88 | 96.26 | 98.60 | 10.59 | 25.21 | 28.01 | 6.18 | 17.30 | 20.52 | 5.09 | 14.43 | 17.49 |
| control | 94.67 | 100.00 | 100.00 | 12.67 | 36.77 | 47.68 | 7.67 | 25.44 | 36.10 | 6.13 | 20.92 | 30.46 |
| processor | 94.00 | 100.00 | 100.00 | 59.00 | 88.89 | 95.26 | 4.00 | 18.95 | 35.26 | 1.55 | 7.72 | 15.32 |
| crypto | 96.25 | 100.00 | 100.00 | 10.00 | 32.96 | 43.94 | 10.00 | 32.96 | 43.94 | 1.47 | 5.40 | 8.22 |

Table 21: Detailed performance results for `Deepseek-Reasoner`. All values in %. Dark gray = Best, Light gray = Second Best.

| Split | syn@1 | syn@5 | syn@10 | fun@1 | fun@5 | fun@10 | opt@1 | opt@5 | opt@10 | eff@1 | eff@5 | eff@10 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|
| **Complexity** | | | | | | | | | | | | |
| 0 | 94.49 | 99.32 | 99.73 | 57.87 | 84.84 | 90.57 | 27.47 | 50.72 | 60.48 | 24.19 | 45.44 | 54.84 |
| 1 | 92.99 | 99.85 | 100.00 | 45.53 | 76.36 | 85.13 | 29.89 | 59.93 | 71.52 | 24.73 | 52.15 | 64.03 |
| 2 | 87.25 | 97.19 | 98.73 | 41.08 | 61.09 | 66.36 | 23.53 | 50.79 | 60.23 | 15.77 | 40.13 | 50.76 |
| 3 | 81.67 | 99.48 | 100.00 | 15.83 | 49.77 | 63.46 | 10.83 | 34.16 | 48.24 | 10.76 | 34.12 | 48.24 |
| **Category** | | | | | | | | | | | | |
| arithmetic | 92.67 | 99.29 | 99.72 | 53.60 | 83.26 | 88.65 | 23.37 | 47.27 | 56.85 | 20.45 | 41.65 | 50.34 |
| dsp | 89.30 | 99.71 | 100.00 | 44.00 | 80.10 | 89.39 | 29.90 | 61.66 | 73.36 | 26.06 | 54.52 | 66.79 |
| communication | 94.06 | 99.83 | 100.00 | 50.73 | 72.42 | 78.88 | 32.19 | 63.97 | 73.43 | 24.03 | 52.39 | 62.97 |
| miscellaneous | 94.87 | 99.99 | 100.00 | 51.58 | 76.45 | 85.56 | 36.84 | 63.68 | 74.76 | 31.12 | 56.58 | 67.87 |
| datapath | 88.00 | 97.28 | 99.41 | 47.00 | 68.04 | 74.10 | 28.20 | 48.56 | 57.75 | 21.26 | 41.16 | 50.92 |
| memory | 90.00 | 95.59 | 97.06 | 24.71 | 53.00 | 65.23 | 14.71 | 42.11 | 55.21 | 12.18 | 36.63 | 50.15 |
| control | 95.67 | 100.00 | 100.00 | 30.00 | 62.62 | 71.42 | 21.00 | 51.69 | 64.51 | 17.28 | 48.08 | 63.01 |
| processor | 94.00 | 100.00 | 100.00 | 73.00 | 99.08 | 100.00 | 13.00 | 50.16 | 75.46 | 10.85 | 44.66 | 71.46 |
| crypto | 91.25 | 100.00 | 100.00 | 47.50 | 56.25 | 62.50 | 42.50 | 56.25 | 62.50 | 21.19 | 33.00 | 38.29 |

Table 22: Detailed performance results for `Deepseek-Coder-6.7B`. All values in %. Dark gray = Best, Light gray = Second Best.

| Split | syn@1 | syn@5 | syn@10 | fun@1 | fun@5 | fun@10 | opt@1 | opt@5 | opt@10 | eff@1 | eff@5 | eff@10 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|
| **Complexity** | | | | | | | | | | | | |
| 0 | 85.96 | 99.67 | 99.99 | 23.65 | 60.43 | 75.96 | 4.16 | 14.98 | 22.82 | 3.23 | 12.26 | 18.81 |
| 1 | 78.98 | 97.94 | 98.50 | 18.77 | 56.39 | 74.96 | 3.42 | 13.23 | 20.40 | 2.51 | 10.21 | 16.42 |
| 2 | 60.69 | 91.20 | 95.41 | 15.29 | 50.31 | 68.85 | 5.10 | 17.71 | 24.84 | 3.60 | 14.07 | 21.82 |
| 3 | 25.83 | 56.48 | 64.80 | 5.83 | 15.28 | 16.64 | 0.83 | 4.17 | 8.33 | 0.83 | 4.17 | 8.33 |
| **Category** | | | | | | | | | | | | |
| arithmetic | 86.34 | 99.87 | 100.00 | 22.56 | 62.41 | 79.58 | 2.97 | 12.51 | 20.52 | 2.39 | 10.13 | 16.74 |
| dsp | 72.20 | 95.51 | 96.96 | 14.20 | 45.99 | 65.01 | 2.10 | 7.69 | 11.40 | 1.93 | 6.90 | 9.96 |
| communication | 59.79 | 90.43 | 94.11 | 14.69 | 46.52 | 62.60 | 6.46 | 22.36 | 31.35 | 3.99 | 15.80 | 24.44 |
| miscellaneous | 84.74 | 99.97 | 100.00 | 84.74 | 99.97 | 100.00 | 5.66 | 21.86 | 33.44 | 4.78 | 18.94 | 30.01 |
| datapath | 75.60 | 95.23 | 95.98 | 23.40 | 58.47 | 74.63 | 5.20 | 14.39 | 18.88 | 3.37 | 11.22 | 15.87 |
| memory | 65.29 | 88.12 | 92.78 | 13.82 | 46.85 | 65.97 | 0.59 | 2.94 | 5.88 | 0.59 | 2.94 | 5.88 |
| control | 80.67 | 99.11 | 99.96 | 19.67 | 55.20 | 70.68 | 3.67 | 14.03 | 20.94 | 3.25 | 13.01 | 20.10 |
| processor | 89.00 | 100.00 | 100.00 | 36.00 | 85.82 | 97.54 | 0.00 | 0.00 | 0.00 | 0.00 | 0.00 | 0.00 |
| crypto | 83.75 | 99.97 | 100.00 | 22.50 | 72.22 | 92.55 | 10.00 | 38.91 | 62.09 | 3.39 | 15.72 | 29.73 |

Table 23: Detailed performance results for `CodeLlama-7B-Instruct`. All values in %. Dark gray = Best, Light gray = Second Best.

| Split | syn@1 | syn@5 | syn@10 | fun@1 | fun@5 | fun@10 | opt@1 | opt@5 | opt@10 | eff@1 | eff@5 | eff@10 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|
| **Complexity** | | | | | | | | | | | | |
| 0 | 78.09 | 99.76 | 99.99 | 50.84 | 94.08 | 99.16 | 2.64 | 11.34 | 19.32 | 1.96 | 8.68 | 15.17 |
| 1 | 73.10 | 97.53 | 98.82 | 50.07 | 90.05 | 95.84 | 2.54 | 9.60 | 15.99 | 1.76 | 6.47 | 10.92 |
| 2 | 55.10 | 89.66 | 95.08 | 38.73 | 76.98 | 84.82 | 2.16 | 8.93 | 14.57 | 0.96 | 4.35 | 7.77 |
| 3 | 35.83 | 61.48 | 71.05 | 14.17 | 30.83 | 39.91 | 1.67 | 8.33 | 16.67 | 1.36 | 6.82 | 13.65 |
| **Category** | | | | | | | | | | | | |
| arithmetic | 77.85 | 99.70 | 99.99 | 51.40 | 94.64 | 99.25 | 2.15 | 9.27 | 16.04 | 1.52 | 6.81 | 12.14 |
| dsp | 71.50 | 96.96 | 99.03 | 47.20 | 84.70 | 91.29 | 2.20 | 6.48 | 9.84 | 1.67 | 4.31 | 6.50 |
| communication | 52.71 | 88.42 | 93.38 | 37.40 | 79.82 | 87.54 | 3.33 | 13.48 | 22.72 | 1.81 | 7.44 | 12.95 |
| miscellaneous | 72.76 | 97.59 | 99.73 | 50.00 | 89.46 | 96.67 | 1.32 | 6.05 | 10.94 | 1.01 | 4.57 | 8.13 |
| datapath | 81.60 | 98.37 | 99.58 | 55.80 | 93.24 | 95.80 | 3.80 | 15.68 | 25.87 | 3.29 | 13.97 | 23.57 |
| memory | 62.06 | 85.98 | 87.97 | 44.12 | 76.39 | 81.37 | 3.82 | 16.27 | 27.14 | 1.95 | 9.07 | 16.62 |
| control | 70.00 | 98.37 | 99.96 | 56.67 | 93.17 | 98.89 | 2.00 | 9.30 | 16.84 | 1.25 | 5.80 | 10.42 |
| processor | 74.00 | 99.94 | 100.00 | 17.00 | 55.43 | 72.80 | 0.00 | 0.00 | 0.00 | 0.00 | 0.00 | 0.00 |
| crypto | 68.75 | 99.57 | 100.00 | 35.00 | 90.26 | 99.51 | 5.00 | 17.96 | 23.92 | 1.95 | 8.00 | 12.56 |

Table 24: Detailed performance results for `CodeQwen1.5-7B-Chat`. All values in %. Dark gray = Best, Light gray = Second Best.

| Split | syn@1 | syn@5 | syn@10 | fun@1 | fun@5 | fun@10 | opt@1 | opt@5 | opt@10 | eff@1 | eff@5 | eff@10 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|
| **Complexity** | | | | | | | | | | | | |
| 0 | 81.85 | 99.80 | 99.99 | 27.53 | 71.14 | 87.22 | 6.07 | 22.23 | 33.47 | 4.94 | 17.98 | 27.01 |
| 1 | 72.25 | 98.63 | 99.29 | 15.70 | 48.96 | 66.70 | 4.37 | 15.70 | 24.31 | 2.91 | 11.06 | 17.50 |
| 2 | 57.25 | 92.44 | 97.36 | 15.49 | 45.61 | 59.12 | 3.33 | 14.25 | 24.08 | 2.81 | 11.88 | 19.82 |
| 3 | 31.67 | 62.77 | 74.01 | 11.67 | 20.81 | 25.00 | 0.83 | 4.17 | 8.33 | 0.83 | 4.17 | 8.33 |
| **Category** | | | | | | | | | | | | |
| arithmetic | 79.24 | 99.28 | 99.94 | 24.71 | 68.67 | 85.78 | 4.88 | 18.53 | 28.98 | 3.63 | 13.55 | 21.04 |
| dsp | 70.20 | 96.43 | 97.99 | 13.40 | 42.48 | 60.51 | 2.70 | 11.76 | 20.20 | 1.83 | 7.94 | 13.52 |
| communication | 52.71 | 93.82 | 98.32 | 13.23 | 45.70 | 63.86 | 3.54 | 15.50 | 26.78 | 2.88 | 12.57 | 21.79 |
| miscellaneous | 81.45 | 99.98 | 100.00 | 18.68 | 51.40 | 67.46 | 4.74 | 18.20 | 27.93 | 3.62 | 13.54 | 20.53 |
| datapath | 77.40 | 99.82 | 100.00 | 23.20 | 58.94 | 72.43 | 6.80 | 23.32 | 32.63 | 5.70 | 20.88 | 30.53 |
| memory | 66.47 | 87.55 | 88.23 | 21.76 | 53.22 | 63.91 | 7.65 | 20.60 | 27.40 | 3.97 | 14.42 | 22.39 |
| control | 61.00 | 95.50 | 99.57 | 16.00 | 39.05 | 48.30 | 7.00 | 19.92 | 25.95 | 6.18 | 18.15 | 23.88 |
| processor | 79.00 | 99.95 | 100.00 | 16.00 | 53.67 | 72.26 | 1.00 | 5.00 | 10.00 | 0.03 | 0.16 | 0.32 |
| crypto | 82.50 | 100.00 | 100.00 | 32.50 | 81.50 | 96.28 | 5.00 | 21.27 | 34.87 | 4.73 | 20.34 | 33.82 |

Table 25: Detailed performance results for `CodeV-DS-6.7B`. All values in %. Dark gray = Best, Light gray = Second Best.

| Split | syn@1 | syn@5 | syn@10 | fun@1 | fun@5 | fun@10 | opt@1 | opt@5 | opt@10 | eff@1 | eff@5 | eff@10 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|
| **Complexity** | | | | | | | | | | | | |
| 0 | 10.06 | 30.16 | 45.76 | 3.31 | 15.09 | 27.16 | 0.11 | 0.56 | 1.12 | 0.09 | 0.45 | 0.91 |
| 1 | 7.25 | 22.85 | 34.55 | 3.17 | 13.10 | 22.05 | 0.28 | 1.21 | 2.08 | 0.19 | 0.88 | 1.66 |
| 2 | 9.41 | 32.20 | 48.66 | 5.39 | 22.33 | 36.49 | 0.59 | 2.94 | 5.88 | 0.32 | 1.58 | 3.15 |
| 3 | 0.83 | 4.17 | 8.33 | 0.83 | 4.17 | 8.33 | 0.00 | 0.00 | 0.00 | 0.00 | 0.00 | 0.00 |
| **Category** | | | | | | | | | | | | |
| arithmetic | 11.57 | 34.46 | 51.80 | 3.72 | 16.73 | 30.29 | 0.41 | 1.71 | 2.86 | 0.26 | 1.20 | 2.24 |
| dsp | 7.90 | 21.82 | 31.11 | 3.20 | 12.68 | 20.24 | 0.00 | 0.00 | 0.00 | 0.00 | 0.00 | 0.00 |
| communication | 9.79 | 34.07 | 48.27 | 6.25 | 23.99 | 36.46 | 0.52 | 2.60 | 5.21 | 0.32 | 1.62 | 3.23 |
| miscellaneous | 3.82 | 17.46 | 31.23 | 2.63 | 12.47 | 23.20 | 0.26 | 1.32 | 2.63 | 0.14 | 0.68 | 1.36 |
| datapath | 11.80 | 29.51 | 43.50 | 3.20 | 14.39 | 25.26 | 0.00 | 0.00 | 0.00 | 0.00 | 0.00 | 0.00 |
| memory | 4.41 | 19.38 | 32.82 | 2.35 | 10.84 | 19.35 | 0.29 | 1.47 | 2.94 | 0.28 | 1.38 | 2.76 |
| control | 1.67 | 8.33 | 16.67 | 1.00 | 5.00 | 10.00 | 0.00 | 0.00 | 0.00 | 0.00 | 0.00 | 0.00 |
| processor | 1.00 | 5.00 | 10.00 | 1.00 | 5.00 | 10.00 | 0.00 | 0.00 | 0.00 | 0.04 | 0.19 | 0.37 |
| crypto | 1.25 | 6.25 | 12.50 | 1.25 | 6.25 | 12.50 | 1.25 | 6.25 | 12.50 | 0.77 | 3.85 | 7.71 |

Table 26: Detailed performance results for `CodeV-CL-7B`. All values in %. Dark gray = Best, Light gray = Second Best.

| Split | syn@1 | syn@5 | syn@10 | fun@1 | fun@5 | fun@10 | opt@1 | opt@5 | opt@10 | eff@1 | eff@5 | eff@10 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|
| **Complexity** | | | | | | | | | | | | |
| 0 | 21.24 | 64.34 | 83.89 | 13.76 | 49.06 | 70.57 | 0.39 | 1.91 | 3.67 | 0.31 | 1.50 | 2.85 |
| 1 | 17.46 | 58.21 | 79.17 | 12.54 | 45.61 | 66.02 | 0.85 | 3.81 | 6.79 | 0.54 | 2.43 | 4.32 |
| 2 | 15.29 | 51.85 | 70.42 | 10.49 | 38.44 | 54.93 | 2.16 | 8.83 | 14.24 | 1.17 | 4.99 | 8.42 |
| 3 | 7.50 | 22.74 | 29.36 | 5.83 | 15.28 | 16.64 | 0.83 | 4.17 | 8.33 | 0.83 | 4.17 | 8.33 |
| **Category** | | | | | | | | | | | | |
| arithmetic | 18.72 | 59.69 | 81.01 | 11.80 | 43.83 | 65.84 | 0.35 | 1.74 | 3.49 | 0.22 | 1.09 | 2.19 |
| dsp | 16.30 | 53.41 | 71.50 | 10.80 | 39.01 | 55.75 | 0.70 | 3.29 | 6.05 | 0.43 | 2.02 | 3.70 |
| communication | 13.65 | 49.84 | 71.55 | 10.62 | 41.00 | 61.30 | 2.81 | 11.40 | 18.17 | 1.80 | 7.59 | 12.67 |
| miscellaneous | 22.63 | 68.84 | 86.10 | 17.50 | 58.02 | 77.28 | 0.39 | 1.84 | 3.32 | 0.39 | 1.84 | 3.32 |
| datapath | 18.80 | 57.72 | 79.39 | 11.40 | 42.25 | 62.97 | 0.60 | 3.00 | 6.00 | 0.50 | 2.49 | 4.98 |
| memory | 13.24 | 45.27 | 61.55 | 8.24 | 32.36 | 49.45 | 0.88 | 4.10 | 7.43 | 0.51 | 2.35 | 4.22 |
| control | 19.67 | 65.92 | 87.45 | 16.00 | 55.80 | 75.45 | 1.33 | 5.67 | 9.30 | 0.40 | 1.63 | 2.50 |
| processor | 26.00 | 81.08 | 98.13 | 15.00 | 47.91 | 58.70 | 0.00 | 0.00 | 0.00 | 0.00 | 0.00 | 0.00 |
| crypto | 33.75 | 88.33 | 98.86 | 26.25 | 79.88 | 96.79 | 1.25 | 6.25 | 12.50 | 0.14 | 0.68 | 1.36 |

Table 27: Detailed performance results for `CodeV-QW-7B`. All values in %. Dark gray = Best, Light gray = Second Best.

| Split | syn@1 | syn@5 | syn@10 | fun@1 | fun@5 | fun@10 | opt@1 | opt@5 | opt@10 | eff@1 | eff@5 | eff@10 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|
| **Complexity** | | | | | | | | | | | | |
| 0 | 7.98 | 23.16 | 35.78 | 2.08 | 9.34 | 17.05 | 0.22 | 1.06 | 1.98 | 0.16 | 0.80 | 1.58 |
| 1 | 5.81 | 17.52 | 26.78 | 1.62 | 7.44 | 13.52 | 0.25 | 1.16 | 2.13 | 0.21 | 1.01 | 1.89 |
| 2 | 5.78 | 22.35 | 35.60 | 2.55 | 10.62 | 17.43 | 1.18 | 5.19 | 8.93 | 0.11 | 0.55 | 1.09 |
| 3 | 3.33 | 11.97 | 15.94 | 0.00 | 0.00 | 0.00 | 0.00 | 0.00 | 0.00 | 0.00 | 0.00 | 0.00 |
| **Category** | | | | | | | | | | | | |
| arithmetic | 9.77 | 26.61 | 39.05 | 2.21 | 10.01 | 18.38 | 0.52 | 2.43 | 4.41 | 0.40 | 1.92 | 3.59 |
| dsp | 6.20 | 18.04 | 27.88 | 1.10 | 5.20 | 9.79 | 0.20 | 1.00 | 2.00 | 0.10 | 0.52 | 1.05 |
| communication | 7.29 | 29.16 | 45.68 | 4.58 | 19.03 | 30.82 | 1.15 | 5.00 | 8.44 | 0.11 | 0.56 | 1.10 |
| miscellaneous | 2.11 | 10.25 | 19.81 | 1.18 | 5.92 | 11.84 | 0.13 | 0.66 | 1.32 | 0.13 | 0.66 | 1.32 |
| datapath | 7.00 | 16.02 | 21.93 | 0.20 | 1.00 | 2.00 | 0.00 | 0.00 | 0.00 | 0.00 | 0.00 | 0.00 |
| memory | 3.53 | 12.44 | 19.19 | 1.18 | 5.57 | 10.37 | 0.00 | 0.00 | 0.00 | 0.00 | 0.00 | 0.00 |
| control | 2.00 | 9.65 | 18.42 | 0.67 | 3.33 | 6.67 | 0.00 | 0.00 | 0.00 | 0.00 | 0.00 | 0.00 |
| processor | 1.00 | 5.00 | 10.00 | 0.00 | 0.00 | 0.00 | 0.00 | 0.00 | 0.00 | 0.00 | 0.00 | 0.00 |
| crypto | 0.00 | 0.00 | 0.00 | 0.00 | 0.00 | 0.00 | 0.00 | 0.00 | 0.00 | 0.00 | 0.00 | 0.00 |

Table 28: Detailed performance results for `RTLCoder-Deepseek-v1.1`. All values in %. Dark gray = Best, Light gray = Second Best.

| Split | syn@1 | syn@5 | syn@10 | fun@1 | fun@5 | fun@10 | opt@1 | opt@5 | opt@10 | eff@1 | eff@5 | eff@10 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|
| **Complexity** | | | | | | | | | | | | |
| 0 | 84.33 | 99.32 | 99.73 | 59.21 | 94.39 | 98.46 | 2.92 | 11.36 | 19.19 | 2.03 | 8.16 | 14.20 |
| 1 | 77.68 | 98.21 | 98.58 | 45.18 | 88.84 | 95.02 | 3.52 | 12.98 | 20.69 | 2.55 | 9.52 | 15.28 |
| 2 | 61.76 | 90.38 | 94.32 | 38.33 | 81.08 | 88.58 | 2.65 | 10.41 | 16.31 | 1.27 | 5.20 | 8.45 |
| 3 | 29.17 | 50.93 | 58.06 | 20.00 | 42.79 | 48.24 | 0.83 | 4.17 | 8.33 | 0.83 | 4.17 | 8.33 |
| **Category** | | | | | | | | | | | | |
| arithmetic | 83.20 | 99.79 | 99.99 | 83.20 | 99.79 | 99.99 | 2.33 | 9.73 | 16.78 | 1.52 | 6.69 | 12.00 |
| dsp | 74.60 | 95.94 | 96.97 | 38.80 | 83.78 | 92.70 | 2.70 | 10.00 | 15.81 | 2.35 | 8.36 | 12.81 |
| communication | 59.27 | 90.47 | 93.42 | 33.44 | 76.03 | 85.19 | 5.21 | 17.80 | 27.56 | 2.71 | 9.48 | 15.42 |
| miscellaneous | 83.55 | 99.99 | 100.00 | 53.29 | 94.83 | 98.71 | 3.29 | 13.86 | 22.53 | 2.62 | 11.11 | 18.08 |
| datapath | 78.00 | 93.69 | 95.05 | 50.40 | 89.48 | 94.08 | 4.00 | 12.16 | 17.93 | 2.87 | 9.05 | 13.98 |
| memory | 61.47 | 85.74 | 89.77 | 43.24 | 79.06 | 82.15 | 0.88 | 4.41 | 8.82 | 0.43 | 2.15 | 4.30 |
| control | 78.00 | 98.68 | 99.89 | 49.33 | 91.42 | 97.68 | 4.00 | 16.24 | 26.09 | 2.95 | 12.43 | 20.67 |
| processor | 88.00 | 100.00 | 100.00 | 51.00 | 95.31 | 99.67 | 0.00 | 0.00 | 0.00 | 0.00 | 0.00 | 0.00 |
| crypto | 87.50 | 100.00 | 100.00 | 62.50 | 98.60 | 99.99 | 3.75 | 17.43 | 31.58 | 2.64 | 11.87 | 20.44 |