Building Reasoning LLMs for Hardware Design Generation via Function-Aligned Differentiated Revision

Weimin Fu
Kansas State University
Manhattan, USA
weiminf@ksu.edu

Shijie Li
University of Science and Technology of China
Anhui, China
shijie_li@mail.ustc.edu.cn

Kaichen Yang
Michigan Technological University
Houghton, USA
kaicheny@mtu.edu

Xuan (Silvia) Zhang
Northeastern University
Boston, USA
xuan.zhang@northeastern.edu

Yier Jin
University of Science and Technology of China
Anhui, China
jinyier@ustc.edu.cn

Xiaolong Guo
Kansas State University
Manhattan, USA
guoxiaolong@ksu.edu

***Abstract*—Recent advances in large language models have significantly improved the capabilities of programming. While these models excel at generating valid software, applying them to the hardware domain remains challenging due to the intrinsic complexity and strict structural semantics required in hardware design. Current LLM approaches for hardware generation typically focus on direct generation. This often results in hardware implementations with functional errors or structural flaws. To overcome these limitations, we propose a reasoning-enhanced training framework explicitly tailored for hardware generation tasks. Our multi-stage methodology combines systematic dataset curation via compilation filtering (achieving a 100% pass rate compared to $27-44\%$ in existing datasets), Function-Aligned Differentiated Revision for comparative annotation across five RTL-relevant dimensions, supervised fine-tuning using reasoning prompts, and reinforcement learning guided by Verilator Parser. Our experiments show that explicitly incorporating reasoning substantially enhances the structural integrity and functional correctness of generated hardware designs, improving $pass@1$ rates by up to 20% on VerilogEval Human benchmarks and reproducing the "Aha moment", where the model explicitly organizes ideas before generation. Our work demonstrates that smaller, specialized reasoning models (1.5B parameters) can effectively augment larger open-source language models through reasoning transfer.**

***Index Terms*—Reasoning LLMs, Hardware Generation**

I. Introduction

The application of Large Language Models (LLMs) as intelligent agents in hardware design generations has been explored in various practical implementations. Depending on whether the LLM is modified, these approaches are generally categorized into prompt-based [1]–[6] and fine-tuning methods [7]–[10]. The prompt-based approach enables richer prior knowledge and more extensive corner cases. However, its effectiveness depends on the user's expertise, making it challenging. In contrast, fine-tuning methods can generate results without requiring explicit hardware expertise. However, they suffer from limited performance due to the lack of high-quality hardware design datasets and computational constraints requiring smaller models. Specifically, existing hardware datasets show extremely low verification pass rates (27-44%) when filtered through standard EDA tools, leading to models that learn incorrect design patterns.

Meanwhile, state-of-the-art proprietary LLMs such as GPT-4o [11] and Claude 3.7 Sonnet [12] and open-source models such as DeepSeek V3 [13] and Gemma 3 [14] have demonstrated remarkable capabilities in text processing, task comprehension, and perception. Compared to earlier LLMs, these models have significantly larger scales and are trained on vastly larger datasets, leading to substantial performance improvements. Despite their success, the decision-making process of these LLMs largely relies on heuristic-driven reasoning. However, they fall short on tasks requiring deep logical analysis and long-term predictive reasoning. In contrast, reasoning LLMs (o1 [15], o3 [16], DeepSeek R1 [17]) aim to emulate a slower, more deliberate reasoning process. Thus, hardware design generation, which involves complex logic and corner case handling, is well suited to this paradigm.

However, applying reasoning LLMs to hardware design generation still requires addressing two critical yet often-overlooked challenges: (1) constructing high-quality reasoning datasets for hardware generation, and (2) developing appropriate reward functions for reinforcement learning. To tackle these challenges, this paper proposes the development of a **domain-specific reasoning LLM for hardware design generation**. Our proposed framework bridges the gap by incorporating structured reasoning and compiler feedback. A visual comparison of these approaches is provided in Fig. 1, where a more detailed description is given in Section IV. In summary, the contributions of this paper are:

Portions of this work were supported by the National Science Foundation (2340949 and 2419880).

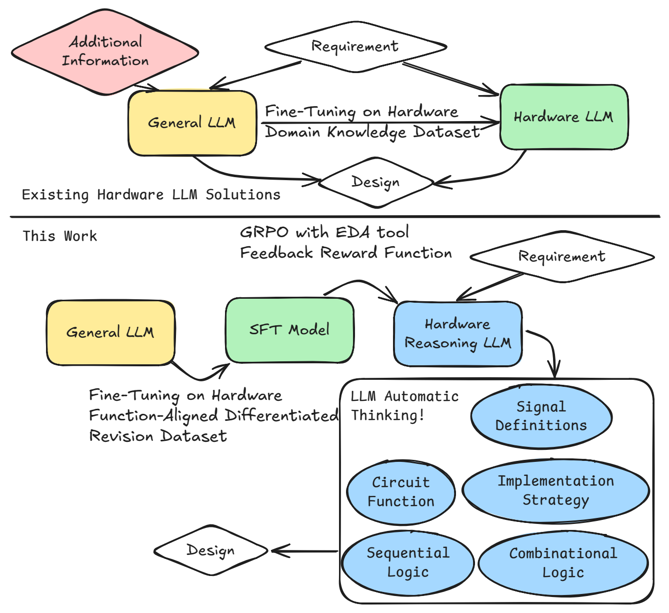


Fig. 1. Comparison of existing hardware-oriented LLM approaches with our proposed reasoning framework. Traditional methods (top) directly map requirements to designs using general LLMs with hardware datasets. Our approach (bottom) introduces structured thinking through Function-Aligned Differentiated Revision and EDA tool feedback, enabling the model to analyze circuit function, implementation strategy, logic organization, and signal definition before generating designs.

- A Function-Aligned Differentiated Revision methodology that builds reasoning datasets by analyzing hardware designs across five RTL-relevant dimensions (Signal Definitions, Combinational Logic, Sequential Logic, Circuit Function, and Implementation Strategy), while leveraging compiler feedback to establish effective reward functions.
- We demonstrate that smaller, specialized reasoning models (1.5B) augment larger open-source language models by enhancing their chain-of-thought capabilities and improving $pass@1$ rates by up to 20% on hardware-specific tasks.
- Our approach reproduces the "Aha moment" effect observed in human hardware designers, where the model explicitly organizes design logic before implementation, showing that well-structured reasoning significantly improves the process and outcomes of hardware generation.

II. Background

A. LLM and Implications for Hardware Tasks

While LLMs have shown impressive results in software domains, their application to hardware design remains limited. Efforts to apply LLMs in the hardware domain fall into two categories: finetuning and prompting. Fine-tuning methods, while promising, are constrained by limited resources, making it difficult to fine-tune larger models. Smaller models, though more accessible, are believed to suffer from lower performance ceilings. Moreover, datasets suitable for training LLMs in the hardware domain are criticized for being too small or lacking quality [18].

Hardware-specific prompting methods rely on either retrieval-augmented generation [19] or multi-turn chain-of-thought prompting integrated with toolchain feedback [20]. However, applying these strategies faces limitations. First, proprietary LLM APIs are infeasible in hardware workflows due to intellectual property sensitivity, which restricts data sharing. Meanwhile, open-source models struggle to support complex prompting reliably, leading to inconsistent performance and limited reproducibility. Second, general-purpose LLMs are not optimized for hardware tasks and frequently degrade performance when applied to structurally constrained domains like RTL. In many cases, their results are comparable to—or even outperformed by—smaller models fine-tuned on hardware-specific datasets [7]. Finally, while prompt engineering with domain heuristics can improve productivity for expert users, it does little to reduce the barrier for non-experts or to automate the reasoning process.

These challenges highlight the need for models that are both reasoning-aware and domain-aligned, eliminating reliance on handcrafted prompts or retrieval-based augmentation. Reasoning LLMs introduce a fundamentally different paradigm: instead of producing outputs immediately, the model first "thinks"—explicitly decomposing the task and organizing its logic—before generating final answers. While this deliberative process increases latency, it significantly improves the performance of complex, structured problems. Recent models such as OpenAI's O1 and DeepSeek R1 have demonstrated that reasoning supervision enables even smaller models to approach the performance of much larger proprietary systems [21]. These models leverage self-reflective capabilities and alignment strategies to overcome scale limitations and generalize effectively.

Bringing this reasoning capability into hardware design tasks, where correctness, modular structure, and optimization constraints are paramount, is beneficial and essential for unlocking the full potential of LLMs in this domain.

B. Hardware Datasets and Verification-Guided Filtering

High-quality training data for hardware-domain LLMs remains scarce. Unlike the software domain, which benefits from large datasets mined from GitHub or Kaggle, hardware code (Verilog, VHDL, SystemVerilog) represents less than 0.1% of major code corpora StarCoder2 [22]. Moreover, these datasets are primarily designed for pretraining, limiting their effectiveness for task-specific fine-tuning. Early hardware-oriented efforts [19], [23], [24] often rely on sparse code from GitHub or textbooks, or version-based diffs [25], resulting in poor coverage and limited task diversity.

A common workaround is to generate synthetic data using LLMs, such as RTLCoder [7], MGVerilog [26], and [27], which pair templated prompts with LLM-generated Verilog. However, general-purpose LLMs produce code with repetitive patterns, syntax errors, and hallucinations, degrading dataset quality and downstream performance. To ensure data quality, we apply Verilator-based filtering to retain only compilable hardware designs [28]. This synthesis-based filtering ensures our corpus is structurally valid and semantically meaningful, improving data quality and reinforcing correctness priors critical for reasoning in hardware generation tasks.

C. Learning Objectives for Reasoning LLMs

Supervised fine-tuning enables LLMs to learn instruction-following behavior from curated input-output pairs. The objective maximizes the likelihood of generating a desired response y given a prompt x:

$$\mathcal{L}_{\text{SFT}}(\phi) = \mathbb{E}_{(x,y)\sim\mathcal{D}_{\text{sft}}}\left[\frac{1}{|y|}\sum_{t=1}^{|y|}\log p_\phi(y_t \mid x, y_{<t})\right] \quad (1)$$

where ϕ represents the model parameters and p_ϕ denotes the conditional language model.

Reinforcement learning enhances model alignment further, particularly on reasoning tasks. We use Group Relative Policy Optimization (GRPO) [29], which uses a normalized reward computed across a batch of sampled responses $\{y^{(i)}\}_{i=1}^{K}$. For each output $y^{(i)}$, a scalar reward $R_i \in \{0,1\}$ is assigned, and token-level advantages are

$$\hat{\mathcal{A}}_{i,t} = \frac{R_i - \mu_R}{\sigma_R}, \quad \text{where} \quad \mu_R = \frac{1}{K}\sum_{j=1}^{K} R_j,$$
$$\sigma_R = \sqrt{\frac{1}{K}\sum_{j=1}^{K}(R_j - \mu_R)^2} \quad (2)$$

The GRPO objective is

$$\mathcal{L}_{\text{GRPO}}(\phi) = \frac{1}{K}\sum_{i=1}^{K}\frac{1}{|y^{(i)}|}\sum_{t=1}^{|y^{(i)}|}$$
$$\left[\frac{p_\phi(y_t^{(i)} \mid x, y_{<t}^{(i)})}{p_{\phi_{\text{old}}}(y_t^{(i)} \mid x, y_{<t}^{(i)})}\hat{\mathcal{A}}_{i,t} - \lambda \cdot \text{KL}(p_\phi \parallel p_{\text{ref}})\right] \quad (3)$$

where p_{ref} is a reference policy (the orignal model) and λ is a coefficient balancing exploration and stability.

In our framework, supervised fine-tuning first establishes core reasoning capabilities from hardware-specific corpora, which are then further refined using GRPO to encourage structurally and semantically correct code generation under reinforcement feedback.

III. Reasoning Hardware LLM and Experimental Results

To develop hardware-specialized reasoning capabilities in large language models, we design a multi-stage training framework consisting of four major components: (1) Verilog-centric dataset construction, (2) functionality-aware comparative annotation via Function-Aligned Differentiated Reasoning, (3) supervised fine-tuning with semantically rich prompts, and (4) reinforcement learning through GRPO, guided by a parser reward function. For clarity and consistency, we present experimental results alongside the description of each corresponding component in this section.

A. Experimental Setup and Environmental Impact Assessment

Our training utilized models from the Qwen2.5 [30](1.5B, 7B, 14B, 32B). All training and inference operations were conducted on a cluster of seven AMD MI210 GPUs. We employed the dataset constructed as described in subsequent sections. We leveraged QWQ-32B [31] for all data synthesis tasks. For the Supervised Fine-Tuning phase, we implemented FlashAttention-2 [32] and Galore [33] optimizations for the 1.5B model, while employing PISSA [34] for non-1.5B models. GRPO loss calculation is based on trl [35]. Cumulative training across all experiments was $1,013$ hours, with estimated energy consumption of $2,553$ kWh, resulting in greenhouse gas emissions (CO_2-equivalent) of approximately 0.88 tonnes.

B. Dataset Curation via Verilog Compilation Filtering

We source open-source RTL hardware designs written in Verilog and SystemVerilog. The raw corpus $\mathcal{D}_{\text{raw}} = \{d_i\}_{i=1}^{N}$ comprises heterogeneous design artifacts with varying structural and syntactic quality. To address the low compilation success rates observed in prior datasets (Section II-B), we apply a strict Verilator-based filtering strategy. Specifically, we retain only those designs that pass both the parsing and elaboration stages of Verilator without errors:

$$\mathcal{D}_{\text{valid}} = \{d_i \in \mathcal{D}_{\text{raw}} \mid \text{Verilator}(d_i) = \texttt{pass}\}$$

This filtering step is essential, as many existing datasets, especially those from LLMs, lack transparency about the model version and often contain non-compilable or semantically invalid code. While modern LLMs still struggle with hardware synthesis, earlier models tend to produce more syntax and logic errors.

To quantify the extent of this issue, we report the Verilator pass rates of several public datasets in Table I.

TABLE I
COMPILATION FILTERING RESULTS USING VERILATOR ACROSS PUBLIC DATASETS

Source	Amount	Parser Pass	Ratio
MGVerilog [26]	$43,792$	$19,411$	44%
RTLCoder [7]	$26,532$	$11,056$	42%
[27]	$68,122$	$18,392$	27%
Our Raw dataset	$48,969$	$48,969$	100%

1) Failure Mode Analysis: To better understand the causes behind compilation failures, we analyze a sample of Verilog files rejected by Verilator. We identify three representative categories that highlight common LLM-specific issues.

- **Hallucinated Modules or Missing Dependencies**: The model instantiates modules that do not exist or are not accompanied.
```
module top; xlslice slice_inst();
endmodule
```
- **Syntax Errors**: Invalid grammar such as misplaced operators or malformed module definitions.
```
module top. endmodule
```

- **Hardware Semantics Violations**: Violations such as assigning to input ports or bit-width mismatches.
  `module top(input wire a); always @(*) a = 1; endmodule`

LLM-generated hardware datasets, while large in scale, often contain syntactic or semantic errors due to hallucination and insufficient domain grounding. When such erroneous data is used for downstream finetuning, it can lead to an unintended ”distillation of errors”—a process in which incorrect design patterns and faulty logic are repeatedly reinforced across training stages. This feedback loop results in a gradual degradation of model quality, ultimately impairing the reasoning and generation capabilities of subsequent models. Therefore, rigorous data filtering is not only beneficial but essential to safeguard the integrity of LLM training pipelines in the hardware domain.

C. *Function-Aligned Differentiated Revision (FADR)*

To support reasoning supervision grounded in hardware semantics, we introduce *Function-Aligned Differentiated Revision (FADR)*. This approach begins by categorizing verified RTL designs into functional groups G_k, each representing a canonical hardware function such as counters, FIFOs, multiplexers, or ALUs. Designs within the same group implement equivalent behavior but differ in structural choices, encoding conventions, and control logic organization.

FADR focuses on systematically comparing design pairs $(d^{(a)}, d^{(b)}) \in G_k$ within the same group to identify and describe their implementation-level differences. These differences often involve rewiring of finite-state machines, changes to reset and enable protocols, or the introduction of logic for corner-case handling. Unlike raw diff-style comparisons, FADR emphasizes semantic relevance and domain-specific abstraction.

Our methodology draws inspiration from the 5W1H approach [18], which introduces structured templates for hardware bug explanation. However, 5W1H relies on journalistic heuristics ill-suited for RTL-level semantics. Instead, FADR reinterprets the idea of structured inquiry by introducing five RTL-specific reasoning dimensions: **Signal Definitions**, **Combinational Logic**, **Sequential Logic**, **Circuit Function**, and **Implementation Strategy**. These dimensions reflect typical axes of divergence among semantically equivalent RTL designs.

For each functional group, we prompt a large language model (QWQ-32B) to conduct pairwise reasoning over verified designs and summarize their differences using the five-dimensional FADR template. This process produces interpretable, structured rationales grounded in hardware engineering intuition. Each resulting instance is encoded as a triplet (q, r, a), where $q = d^{(a)}$ is the original design, r is the five-part FADR reasoning output, and $a = d^{(b)}$ is the target design.

For example, within the “counter” category, FADR may capture a structural difference where one design uses synchronous reset logic (`always @(posedge clk)` with `if (reset)`) and another uses asynchronous reset (`always @(posedge clk or posedge reset_async)`), while also introducing an overflow signal and corresponding combinational detection logic. The FADR output highlights differences in signal declaration, sequential behavior, and high-level design intent.

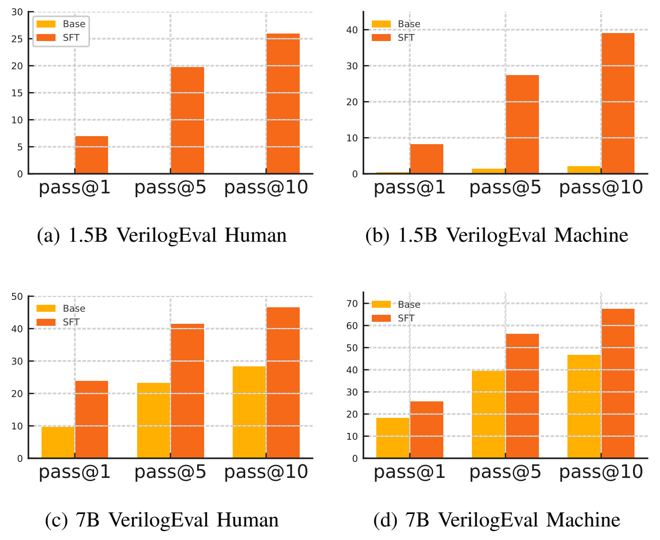
We construct $11,590$ FADR-aligned reasoning samples. While the reasoning content is structured and domain-aware, it is also template-driven and lacks elements of spontaneous self-reflection. This motivates subsequent refinement stages that promote more flexible and introspective reasoning patterns in the model.

D. *Supervised Fine-Tuning with Functional Prompts*

We focus on the task of hardware code generation from functional specifications. Rather than directly mapping a high-level instruction to Verilog code, we introduce a structured intermediate reasoning process to guide generation.

Each training sample consists of three components: (1) **Instruction**, describing the functional goal, (2) **Reasoning**, outlining the design rationale across five RTL aspects from Section III-C and (3) **Answer**, the corresponding Verilog code. This structured reasoning step acts as explicit guidance for code generation. The model is fine-tuned using a standard causal language modeling loss (Eq. 1).

1) Acquiring Hardware Design Capability through Fine-Tuning: Fine-tuning significantly enhances LLMs‘ hardware generation capabilities. As shown in Fig. 2, the 1.5B model exhibits minimal ability to generate hardware code prior to fine-tuning, but demonstrates substantial improvement after training. A similar trend is observed for the 7B model. These results suggest that task-specific fine-tuning consistently improves LLM performance on hardware design tasks.

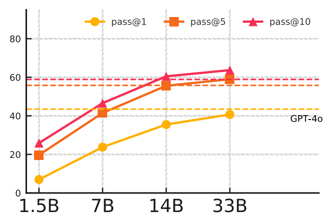


(a) 1.5B VerilogEval Human (b) 1.5B VerilogEval Machine

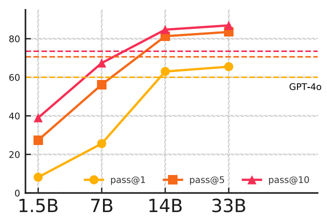
(c) 7B VerilogEval Human (d) 7B VerilogEval Machine

Fig. 2. Fine-tuning with datasets incorporating thinking processes yields significant performance improvements. The orange bars represent the base model—models without fine-tuning—and their performance on the Human and Machine subtasks of VerilogEval. The red bars show the results after fine-tuning with datasets that include thinking processes.

2) Scaling Effects and Observations: From a scaling perspective, while fine-tuning significantly improves model performance, it does not overcome the inherent scaling laws of LLMs. As shown in Fig. 3, performance increases with parameter size under identical fine-tuning conditions. Using proprietary GPT-4o as a reference, models reach parity on the VerilogEval Human benchmark at 33B parameters, and on VerilogEval Machine at 14B. We identify the 14B–33B range as a potential sweet spot for hardware-domain LLMs, where models offer strong performance while remaining feasible for deployment on consumer-grade hardware, enhancing their practical applicability.



(a) VerilogEval Human



(b) VerilogEval Machine

Fig. 3. Scaling Effects of Parameter Size on Hardware Generation Performance with Reasoning-Augmented Fine-Tuning. Reasoning fine-tuning enables near-GPT-4o performance on VerilogEval with mid-size models (14–33B), offering a cost-effective alternative for hardware generation.

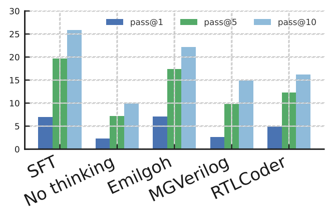
3) Ablation Study: Effectiveness of Reasoning-Augmented Data: We conduct an ablation study to assess the contribution of the *reasoning* component. Specifically, we compare models trained on datasets with and without the reasoning step and models trained on each of the three raw data sources individually. The baseline uses only Verilator-filtered hardware designs paired with functional descriptions, while the reasoning-enhanced dataset includes explicit multi-part explanations that guide the model's generation process.

As shown in Fig. 4, on the more challenging *VerilogEval Human* benchmark, models trained with reasoning consistently outperform not only their non-reasoning counterparts, but also models trained on the three raw datasets. Notably, this improvement is achieved despite the reasoning dataset being only one-third to one-sixth the size of the raw datasets, indicating that the performance gains arise from data quality rather than quantity. This result echoes findings in [36], which show that introducing structured reasoning—even without reinforcement learning—can significantly benefit model performance.

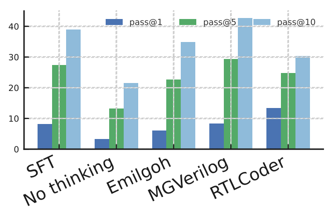
However, the advantage is less pronounced on the *VerilogEval Machine* subset. In some cases, reasoning-trained models perform slightly worse. We attribute this to limitations in the MachineEval benchmark, which its original authors have since deprecated due to concerns about reliability. We include these results for completeness.

E. Reinforcement Learning with Parser Rewards

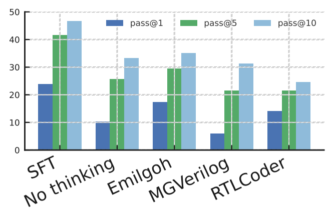
To further refine the model's capacity for producing correct and interpretable outputs, we apply GRPO training with binary reward function. For each prompt, a set of candidate outputs



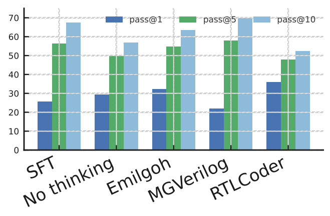
(a) 1.5B VerilogEval Human



(b) 1.5B VerilogEval Machine



(c) 7B VerilogEval Human



(d) 7B VerilogEval Machine

Fig. 4. Ablation study comparing models trained with reasoning-augmented data, non-reasoning data, and three raw datasets. On the VerilogEval Human benchmark, reasoning-trained models consistently outperform all baselines, despite using significantly less data. On the Machine subset, the advantage is less clear, potentially due to known limitations in the benchmark itself.

$\{\hat{y}_i\}_{i=1}^G$ is sampled. Each output is scored using two reward components.

- Format Reward (r_{fmt}): 1 if the output conforms to the required nested format $< thinking > ... < /thinking >< answer > ... < /answer >$; 0 otherwise.
- Syntax Reward (r_{syn}): 1 if the code segment within $< answer > ... < /answer >$ passes Verilator syntax parsing; 0 otherwise.

The total reward is defined as $r_i = r_{\text{fmt}} + r_{\text{syn}}$, and is normalized across the batch to compute the token-wise advantage in Equation 2.

1) Reinforcement Learning with GRPO on 1.5B Model: During the training process, one MI210 GPU is designated as the reference server, responsible for computing the logits of the base SFT model. The remaining six MI210 GPUs perform GRPO-based optimization in parallel. Within each reinforcement learning step, the model generates eight candidate responses per prompt, and their relative quality is compared to compute the policy advantage. Training is optimized using DeepSpeed for efficient multi-GPU communication. In this experiment, we focus on the 1.5B model.

After $1,000$ reinforcement learning steps—corresponding to the first $1,000$ examples in the dataset—we observe notable improvements in the model's generative performance. As shown in Table II, the GRPO-trained model (RL) consistently outperforms the SFT-only baseline across most evaluation metrics. However, it still lags behind larger models such as 7B and beyond, highlighting that RL alone cannot overcome the benefits of scaling.

Interestingly, we observe distinct behavioral patterns in the reasoning-enhanced model. Its failure cases often stem from either (i) overthinking when input information is insufficient, leading to reasoning dead ends, or (ii) weak Verilog coding ability despite correct high-level reasoning.

For example, when designing a 4-bit comparator, the model demonstrates the ”Aha moment” effect, where it first organizes the high-level design structure before proceeding to implementation. The model decomposes the problem into logical steps that mirror human intuition: identifying that a bitwise comparison from MSB to LSB is required, with equality depending on all bits matching and the greater-than condition determined by the first differing bit. The model also strategically selects a parallel combinational logic implementation, demonstrating awareness of hardware constraints like logic depth and timing. This organization of thought before implementation mimics the cognitive process of experienced hardware designers, even though the final code still contained implementation errors.

In another case, after a brief initial analysis, the model becomes trapped in excessive self-questioning and ultimately fails due to perceived input ambiguity.

TABLE II
COMPARISON OF SFT AND GRPO-TRAINED MODELS ON VERILOGEVAL.

Stage	**VerilogEval Human**			**VerilogEval Machine**		
	pass@1	*pass@5*	*pass@10*	*pass@1*	*pass@5*	*pass@10*
SFT	6.96	19.70	25.90	8.18	27.40	39.00
RL(GRPO)	8.00	23.33	40.00	6.00	26.67	42.00

2) Reasoning Transfer: Reinforcement-Trained Prompts for General LLMs: In our final experiment, we investigate whether the reasoning generated by the reinforcement-trained 1.5B model can enhance the performance of a general-purpose LLM. Specifically, we extract the `<thinking>` block from the output and prepend it, along with the original prompt, to the input of the general LLM, Qwen 32B. Since this setup involves two separate model generations, we report only the $pass@1$ performance.

As shown in Fig. 5, this reasoning transfer significantly improves the Qwen-32B model's performance on the benchmark, indicating that the reasoning output effectively serves as a high-quality, structured prompt. This behavior is reminiscent of standard prompting methods; however, in our case, the prompts are automatically learned via reinforcement training. Despite these gains, the performance still falls short of that achieved by models directly fine-tuned on reasoning-augmented datasets. Thus, the RL-trained model acts as an *automatic reasoning prompt generator*, bridging the gap between raw LLMs and reasoning-aware performance.

IV. RELATED WORK

We organize related work according to the architectural comparison in Fig. 1, which highlights the key differences between traditional LLM-based hardware design pipelines (top) and our reasoning-enhanced framework (bottom).

(1) Fine-Tuning with Raw Hardware Datasets:

As shown in the upper right of Fig. 1, many prior works fine-tune LLMs directly on Verilog code paired with functional

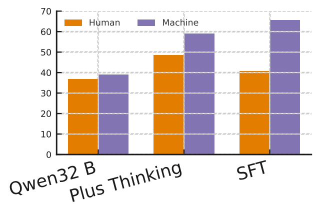


Fig. 5. Injecting `<thinking>` blocks from the GRPO-trained 1.5B model into a general LLM (Qwen-32B) significantly improves pass@1 performance

descriptions, without intermediate reasoning steps. Representative examples include RTLCoder [7], MGVerilog [26], and Emilgoh [27], which rely on large-scale synthetic data. However, these datasets often have low quality and poor verification pass rates. In contrast, our work improves data quality through verilator-based filtering and the building of a reasoning process.

(2) Prompting-Based Approaches:

Prompting methods, exemplified by VerilogReader [6], treat LLMs as passive code readers that generate outputs based on expert-crafted prompts. As shown in the upper left portion of Fig. 1, these approaches bypass intermediate reasoning and depend heavily on prompt engineering. Our framework instead introduces a structured reasoning stage, enabling the model to decompose and plan the design logic before code generation autonomously.

(3) Closed-Source Proprietary Models:

Commercial efforts like NVIDIA's ChipNeMo [37] demonstrate strong performance using proprietary chip design data. However, reproducibility and extensibility remain limited as they are not publicly released. Our approach, shown in the lower part of Fig. 1, complements open-source datasets with a transparent, reasoning-aware training pipeline.

(4) Reasoning-Capable LLMs:

Recent evaluations, such as the study by Collini et al. [38], show that reasoning-oriented models like DeepSeek-R1 and OpenAI's O1 series exhibit structured internal logic when applied to hardware-related tasks. Building on this observation, we adopt reasoning-capable LLMs as the backbone of our framework and incorporate domain-specific supervision via FADR and compiler-guided reinforcement learning. This enables the model to analyze signal definitions, logic structures, and functional intent before generation.

V. DISCUSSION

A. Why do we not compare with proprietary reasoning LLMs?

Too expensive. Reasoning LLMs charge by token, and multi-step reasoning increases tokens per generation. In our experiments, this would raise the cost of a single test pass to a high level, exceeding thousands of USD per batch.

Moreover, preliminary tests using OpenAI's Web interface reveal minimal performance gaps between GPT-4o and O1 on hardware tasks. This is expected, as proprietary models are not explicitly trained with hardware reasoning data or integrated with verification during training. As such, their reasoning behavior in hardware settings does not significantly differ from general-purpose models.

B. Why is Verilator selected as the reward function?

Verilator is widely regarded as a gold standard in the open-source hardware community. Any well-maintained open-source hardware project is expected to pass Verilator simulation, making it a reliable validator. Unlike purely static syntax checkers, Verilator performs executable simulation and produces a strict abstract syntax tree, providing a robust form of syntactic and structural validation. While we do not run full simulations during training, we utilize Verilator to extract ASTs and detect code structure issues. The average runtime is $0.1098s$ per sample, or approximately $1s$ per batch of 8 generations, which is negligible compared to the $150s$ required for a single reinforcement learning step.

Although faster alternatives exist, such as Pyverilog's AST parser, which averages only $0.0016s$ per sample, compatibility remains challenging. In our tests, 92.27% of designs that pass Verilator fail to parse under Pyverilog due to limited syntax support. Incorporating other EDA tools remains a promising direction, but the inherent slowness of hardware simulation may present practical bottlenecks, particularly for non-static, dynamic reward functions.

C. Why is GRPO training performed only on the 1.5B model?

Due to platform constraints, we were limited to training models up to 3B parameters during the reinforcement learning stage. As a result, we conducted complete GRPO training only on the 1.5B model. Nevertheless, all reasoning-fine-tuned models developed in this work will be released, and we encourage the community to further explore reinforcement learning on larger models such as our 32B SFT checkpoint. This opens the door to building even more powerful reasoning-aware hardware generation models.

VI. Conclusion

We presented a reasoning-enhanced training framework for hardware domain reasoning LLM. Our approach significantly improves the' structural correctness and functional fidelity of generated hardware designs, even when using smaller models. Through extensive experiments, we demonstrate that incorporating explicit reasoning enhances standalone performance and is a transferable mechanism for improving general-purpose LLMs. By open-sourcing the models, we aim to lower the barrier to entry for research in reasoning LLM for hardware generation and encourage broader adoption.

Future work includes scaling reinforcement training to larger models, integrating dynamic EDA feedback, and extending reasoning capabilities to broader tasks such as verification and layout generation. We believe reasoning LLMs mark a critical step toward democratizing intelligent hardware design.

References

[1] J. Blocklove, S. Garg *et al.*, "Evaluating llms for hardware design and test," *CoRR*, vol. abs/2405.02326, p. 6, 2024. [Online]. Available: https://doi.org/10.48550/arXiv.2405.02326

[2] Z. Mi, R. Zheng *et al.*, "Promptv: Leveraging llm-powered multi-agent prompting for high-quality verilog generation," *CoRR*, vol. abs/2412.11014, p. 5, 2024. [Online]. Available: https://doi.org/10.48550/arXiv.2412.11014

[3] R. Qiu, G. L. Zhang *et al.*, "Correctbench: Automatic testbench generation with functional self-correction using llms for HDL design," *CoRR*, vol. abs/2411.08510, p. 7, 2024. [Online]. Available: https://doi.org/10.48550/arXiv.2411.08510

[4] K. Thorat, J. Zhao *et al.*, "Advanced large language model (llm)-driven verilog development: Enhancing power, performance, and area optimization in code synthesis," *CoRR*, vol. abs/2312.01022, p. 8, 2023. [Online]. Available: https://doi.org/10.48550/arXiv.2312.01022

[5] Y. Tsai, M. Liu, and H. Ren, "Rtlfixer: Automatically fixing RTL syntax errors with large language models," *CoRR*, vol. abs/2311.16543, p. 7, 2023. [Online]. Available: https://doi.org/10.48550/arXiv.2311.16543

[6] R. Ma, Y. Yang *et al.*, "Verilogreader: Llm-aided hardware test generation," *CoRR*, vol. abs/2406.04373, p. 5, 2024. [Online]. Available: https://doi.org/10.48550/arXiv.2406.04373

[7] S. Liu, W. Fang *et al.*, "Rtlcoder: Fully open-source and efficient llm-assisted rtl code generation technique," *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, vol. Early Access, pp. 1–1, 2024.

[8] F. Cui, C. Yin *et al.*, "Origen:enhancing rtl code generation with code-to-code augmentation and self-reflection," *CoRR*, vol. abs/2407.16237, p. 9, 2024. [Online]. Available: https://doi.org/10.48550/arXiv.2407.16237

[9] Y. Zhao, D. Huang *et al.*, "Codev: Empowering llms for verilog generation through multi-level summarization," *CoRR*, vol. abs/2407.10424, p. 16, 2024. [Online]. Available: https://doi.org/10.48550/arXiv.2407.10424

[10] N. Wang, B. Yao *et al.*, "Large language model for verilog generation with golden code feedback," *CoRR*, vol. abs/2407.18271, p. 9, 2024. [Online]. Available: https://doi.org/10.48550/arXiv.2407.18271

[11] OpenAI, "Chatgpt-4o," https://openai.com/chatgpt, 2024.

[12] Anthropic, "Claude 3.7 sonnet," https://www.anthropic.com/claude, 2025.

[13] DeepSeek-AI, A. Liu *et al.*, "Deepseek-v3 technical report," *CoRR*, vol. abs/2412.19437, 2024. [Online]. Available: https://doi.org/10.48550/arXiv.2412.19437

[14] G. Team, "Gemma 3," 2025. [Online]. Available: https://goo.gle/Gemma3Report

[15] OpenAI, "Learning to reason with LLMs," 9 2024, [Online]. Available: https://openai.com/index/learning-to-reason-with-llms/.

[16] ——, "OpenAI o3-mini," 1 2025, [Online]. Available: https://openai.com/index/openai-o3-mini/.

[17] DeepSeek-AI, D. Guo *et al.*, "Deepseek-r1: Incentivizing reasoning capability in llms via reinforcement learning," *CoRR*, vol. abs/2501.12948, 2025. [Online]. Available: https://doi.org/10.48550/arXiv.2501.12948

[18] W. Fu, S. Li *et al.*, "A generalize hardware debugging approach for large language models semi-synthetic, datasets," *IEEE Transactions on Circuits and Systems I: Regular Papers*, vol. 72, no. 2, pp. 623–636, 2025.

[19] M. Gao, J. Zhao *et al.*, "Autovcoder: A systematic framework for automated verilog code generation using llms," in *42nd IEEE International Conference on Computer Design, ICCD 2024, Milan, Italy, November 18-20, 2024*. IEEE, 2024, pp. 162–169. [Online]. Available: https://doi.org/10.1109/ICCD63220.2024.00033

[20] C. Ho, H. Ren, and B. Khailany, "Verilogcoder: Autonomous verilog coding agents with graph-based planning and abstract syntax tree (ast)-based waveform tracing tool," in *AAAI-25, Sponsored by the Association for the Advancement of Artificial Intelligence, February 25 - March 4, 2025, Philadelphia, PA, USA*, T. Walsh, J. Shah, and Z. Kolter, Eds. AAAI Press, 2025, pp. 300–307. [Online]. Available: https://doi.org/10.1609/aaai.v39i1.32007

[21] N. Team, "Sky-t1: Train your own o1 preview model within $450," https://novasky-ai.github.io/posts/sky-t1, 2025, accessed: 2025-01-09.

[22] A. Lozhkov, R. Li *et al.*, "Starcoder 2 and the stack v2: The next generation," *arXiv preprint arXiv:2402.19173*, 2024.

[23] Z. Pei, H. Zhen *et al.*, "Betterv: Controlled verilog generation with discriminative guidance," in *Forty-first International Conference on Machine Learning, ICML 2024, Vienna, Austria, July 21-27, 2024*. OpenReview.net, 2024. [Online]. Available: https://openreview.net/forum?id=jKnW7r7de1

[24] K. Xu, J. Sun *et al.*, "MEIC: re-thinking RTL debug automation using llms," in *Proceedings of the 43rd IEEE/ACM International Conference on Computer-Aided Design, ICCAD 2024, Newark Liberty International Airport Marriott, NJ, USA, October 27-31, 2024*, J. Xiong and R. Wille, Eds. ACM, 2024, pp. 100:1–100:9. [Online]. Available: https://doi.org/10.1145/3676536.3676801

[25] W. Fu, K. Yang *et al.*, "Llm4sechw: Leveraging domain specific large language model for hardware debugging," *CoRR*, vol. abs/2401.16448, 2024. [Online]. Available: https://doi.org/10.48550/arXiv.2401.16448

[26] Y. Zhang, Z. Yu *et al.*, "Mg-verilog: Multi-grained dataset towards enhanced llm-assisted verilog generation," *CoRR*, vol. abs/2407.01910, 2024. [Online]. Available: https://doi.org/10.48550/arXiv.2407.01910

[27] E. Goh, M. Xiang *et al.*, "From english to ASIC: hardware implementation with large language model," *CoRR*, vol. abs/2403.07039, 2024. [Online]. Available: https://doi.org/10.48550/arXiv.2403.07039

[28] W. Snyder, "Verilator: Open simulation-growing up," *DVClub Bristol*, 2013.

[29] Z. Shao, P. Wang *et al.*, "Deepseekmath: Pushing the limits of mathematical reasoning in open language models," *CoRR*, vol. abs/2402.03300, 2024. [Online]. Available: https://doi.org/10.48550/arXiv.2402.03300

[30] A. Yang, B. Yang *et al.*, "Qwen2.5 technical report," *arXiv preprint arXiv:2412.15115*, 2024.

[31] Q. Team, "Qwq-32b: Embracing the power of reinforcement learning," March 2025. [Online]. Available: https://qwenlm.github.io/blog/qwq-32b/

[32] T. Dao, "Flashattention-2: Faster attention with better parallelism and work partitioning," in *The Twelfth International Conference on Learning Representations, ICLR 2024, Vienna, Austria, May 7-11, 2024*. OpenReview.net, 2024. [Online]. Available: https://openreview.net/forum?id=mZn2Xyh9Ec

[33] J. Zhao, Z. Zhang *et al.*, "Galore: Memory-efficient LLM training by gradient low-rank projection," in *Forty-first International Conference on Machine Learning, ICML 2024, Vienna, Austria, July 21-27, 2024*. OpenReview.net, 2024. [Online]. Available: https://openreview.net/forum?id=hYHsrKDiX7

[34] F. Meng, Z. Wang, and M. Zhang, "Pissa: Principal singular values and singular vectors adaptation of large language models," in *Advances in Neural Information Processing Systems 38: Annual Conference on Neural Information Processing Systems 2024, NeurIPS 2024, Vancouver, BC, Canada, December 10 - 15, 2024*, A. Globersons, L. Mackey *et al.*, Eds., 2024. [Online]. Available: http://papers.nips.cc/paper_files/paper/2024/hash/db36f4d603cc9e3a2a5e10b93e6428f2-Abstract-Conference.html

[35] L. von Werra, Y. Belkada *et al.*, "TRL: Transformer Reinforcement Learning," https://github.com/huggingface/trl, 2023, if you use this software, please cite it using the metadata from this file.

[36] K. Gandhi, A. Chakravarthy *et al.*, "Cognitive behaviors that enable self-improving reasoners, or, four habits of highly effective stars," *CoRR*, vol. abs/2503.01307, 2025. [Online]. Available: https://doi.org/10.48550/arXiv.2503.01307

[37] M. Liu, T.-D. Ene *et al.*, "Chipnemo: Domain-adapted llms for chip design," *arXiv preprint arXiv:2311.00176*, 2023.

[38] L. Collini, A. Hennessee *et al.*, "Can reasoning models reason about hardware? an agentic hls perspective," *arXiv preprint arXiv:2503.12721*, 2025.