EARL: Entropy-Aware RL Alignment of LLMs for Reliable RTL Code Generation

Jiahe Shi
MIT

Zhengqi Gao
MIT

Ching-Yun Ko
IBM

Duane Boning
MIT

***Abstract*—Recent advances in large language models (LLMs) have demonstrated significant potential in hardware design automation, particularly in using natural language to synthesize Register-Transfer Level (RTL) code. Despite this progress, a gap emains between model capability and the demands of real-world TL design, including syntax errors, functional hallucinations, nd weak alignment to designer intent. Reinforcement Learning ith Verifiable Rewards (RLVR) offers a promising approach to ridge this gap, as hardware provides executable and formally heckable signals that can be used to further align model utputs with design intent. However, in long, structured RTL ode sequences, not all tokens contribute equally to functional orrectness, and naïvely spreading gradients across all tokens ilutes learning signals. A key insight from our entropy analysis n RTL generation is that only a small fraction of tokens (e.g., `lways`, `if`, `assign`, `posedge`) exhibit high uncertainty and argely influence control flow and module structure. To address hese challenges, we present EARL, an Entropy-Aware Reinforcee-ent Learning framework for Verilog generation. EARL performs olicy optimization using verifiable reward signals and introduces ntropy-guided selective updates that gate policy gradients to igh-entropy tokens. This approach preserves training stability nd concentrates gradient updates on functionally important egions of code. Our experiments on VerilogEval and RTLLM how that EARL improves functional pass rates over prior LLM aselines by up to 14.7%, while reducing unnecessary updates and mproving training stability. These results indicate that focusing L on critical, high-uncertainty tokens enables more reliable and argeted policy improvement for structured RTL code generation.**

I. Introduction

Large Language Models (LLMs) have achieved remarkable uccess in natural language processing and general-purpose ode generation, demonstrating strong capabilities in undertanding context, modeling semantics, and synthesizing coherent outputs across a wide range of tasks [1], [2]. While these models have revolutionized software development workflows, applying LLMs to specialized domains such as hardware design remains a challenging frontier [3].

One such challenge is the generation of Register-Transfer Level (RTL) code, typically written in Verilog. RTL code lies as the foundation for digital hardware design, translating architectural intent into synthesizable logic. Writing RTL code requires deep domain knowledge and significant engineering effort. Small errors in syntax, port declarations, or timing semantics can lead to non-compilable or functionally incorrect designs. Automating RTL code generation has the potential to significantly accelerate hardware development cycles, reduce engineering effort, and make hardware design more accessible.

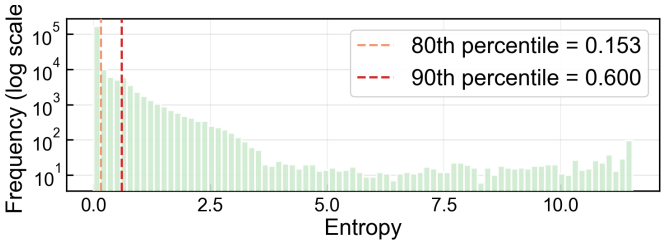


Fig. 1: Token-level entropy distribution in RTL generation.

Recent efforts have explored applying LLMs to Verilog generation, with promising results in syntactic fluency and basic design translation [4]–[6].

However, achieving reliable RTL generation remains challenging. Most existing approaches rely on supervised fine-tuning (SFT) on pre-collected datasets [7], [8]. While this improves syntactic fluency, compiler and testbench feedback, which is essential for functional correctness, remains unused during training. This motivates reinforcement learning with verifiable rewards (RLVR) [9], which leverages executable and formally checkable signals to align model outputs with design intent. Yet, applying RLVR to RTL code generation introduces two fundamental difficulties. First, compiler and testbench feedback is inherently sparse and delayed, making credit assignment particularly challenging in long, structured code sequences [10]. Second, standard RLVR methods spread updates uniformly across all tokens, even though only a small subset governs structural and functional correctness. These limitations highlight the need for entropy-aware reinforcement learning methods that selectively emphasize critical tokens while preserving stability on deterministic ones.

Addressing these challenges requires understanding how learning signals distribute across tokens in RTL sequences. To this end, we conduct an entropy analysis of RTL code generation, which further highlights the limitations of uniform updates. As shown in Fig. 1, the entropy distribution is highly skewed. Most tokens are generated with near-zero entropy, while only a small fraction exhibit significantly higher uncertainty. This pattern echoes recent findings in RLVR research [10]–[12]. In RTL, the imbalance is even more pronounced, since deterministic syntax and port declarations dominate sequences, while high-entropy tokens correspond to module structure, control flow, and signal connections that largely determine correctness. These findings motivate the need for reinforcement learning updates that focus on critical, high-entropy tokens rather than treating all positions uniformly. We

provide a more detailed entropy analysis in RTL generation in Section III.

To address these challenges, we propose **EARL**, an *Entropy-Aware Reinforcement Learning* framework for Verilog generation. EARL builds on supervised initialization and utlizes verifiable reward signals, e.g., syntax validity, interface compliance, and functional correctness. Crucially, EARL introduces entropy-guided selective updates. Instead of spreading gradients uniformly across all tokens, it concentrates policy updates on high-entropy tokens that disproportionately determine module structure, control flow, and downstream correctness. By concentrating policy gradients on uncertain and functionally critical tokens, EARL improves alignment with hardware correctness while preserving training stability. Our contributions are summarized as follows:

1) **Token-level entropy analysis for RTL.** We provide the first entropy study in RTL generation. Our analysis shows that while most tokens are deterministic, a small fraction of high-entropy tokens significantly contribute to functional correctness. This motivates the need for reinforcement learning methods that update policies selectively rather than uniformly.
2) **Entropy-aware RL framework.** We introduce EARL, an entropy-aware RL framework that integrates verifiable rewards with selective policy updates on high-entropy tokens. Unlike uniform update schemes, EARL targets high-entropy tokens while preserving coherence on low-entropy ones, improving alignment with hardware correctness and stabilizing training dynamics. The design is optimizer-agnostic and can be layered on standard objectives. We instantiate it with both PPO and DAPO in this work.
3) **Verifiable multi-signal reward aligned with RTL practice.** We combine compiler validity, interface consistency, and testbench outcomes into a verifiable reward that reflects hardware design constraints. This multi-signal feedback provides executable and formally checkable guidance that goes beyond textual fluency, aligning generation with downstream RTL verification requirements.
4) **Verilog code generation performance improvement.** EARL consistently improves the functional correctness of LLMs on standard RTL benchmarks. Without relying on task-specific prompting or iterative repair, EARL surpasses strong supervised and RL baselines, achieving up to **14.7%** increase in pass@5 on RTLLM.

II. Background

A. Large Language Model for Verilog Code Generation

Recent progress in large language models (LLMs) has spurred research into thier application to Verilog code generation [4], [7], [8], [13]–[18]. Existing approaches can be grouped into three categories: *supervised fine-tuning*, *prompt engineering*, and *reinforcement learning*.

Supervised fine-tuning (SFT) methods adapt pre-trained LLMs using hardware-specific datasets. Examples include RTLCoder [8] and VeriGen [13], which fine-tune on automatically generated instruction–code pairs. Other works enrich data through domain-specific transformations or augmentations. For instance, BetterV [17] leverages Verilog-to-C translation. Despite their gains, SFT methods are constrained by limited and noisy datasets and remain static, lacking adaptive feedback.

Prompt engineering approaches guide models via specialized prompting or interactive tool calls. RTLFixer [15] employs ReAct prompting and retrieval to iteratively fix syntax errors. OriGen [7] combines exemplar prompting with self-reflection and code-to-code transformations, while HaVen [18] introduces a hallucination taxonomy and employs chain-of-thought prompting over symbolic modalities. Prompt-based methods can improve syntactic and functional correctness, but often suffer from high computational cost and poor scalability due to reliance on repeated synthesis or tool feedback.

Reinforcement learning (RL) has recently been explored to bridge the gap between fluency and functional correctness. Building on RLHF [19], [20], methods like PPOCoder [21] and PLUM [22] leverage compiler or test outcomes as reward signals. For Verilog generation, RLVR-style methods have emerged [23], [24], but existing designs are limited: rewards based on AST similarity can be gamed without functional benefit [23], while structured reasoning with distillation may constrain diversity [24]. These limitations highlight the need for reinforcement learning frameworks that leverage verifiable multi-signal feedback while avoiding uniform or inefficient token-level updates.

B. RLVR Algorithms

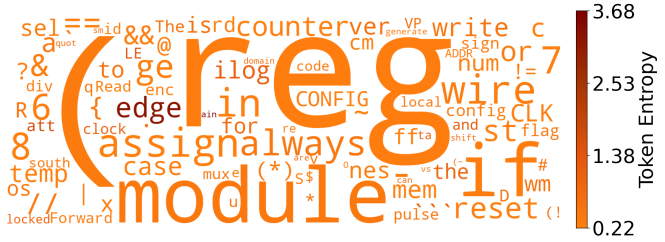
Proximal Policy Optimization (PPO). PPO [25] is a widely used policy gradient algorithm that stabilizes reinforcement learning by clipping the probability ratio between new policy π_θ and old policy $\pi_{\theta_{\text{old}}}$. The objective is as follows:

$$J_{\text{PPO}}(\theta) = \mathbb{E}_{(q,a)\sim\mathcal{D},\, o\sim\pi_{\theta_{\text{old}}}(\cdot|q)}\Big[\min\big(r_t(\theta)\,\hat{A}_t,\; \text{clip}\big(r_t(\theta),\,1-\epsilon,\,1+\epsilon\big)\,\hat{A}_t\big)\Big],$$
$$\text{with} \qquad r_t(\theta) = \frac{\pi_\theta(o_t \mid q,\, o_{<t})}{\pi_{\theta_{\text{old}}}(o_t \mid q,\, o_{<t})},$$

where $\mathcal{D}$ is a dataset of queries q and corresponding ground-truth answers a. $\mathbf{o} = (o_1, \ldots, o_T)$ denotes a sampled output sequence from the old policy $\pi_{\theta_{\text{old}}}$, $\epsilon \in \mathbb{R}$ is a clipping hyperparameter, and $\hat{A}_t$ is the estimated advatage computed using a value network.

Group Relative Policy Optimization (GRPO). GRPO [26] is a variant of PPO designed to reduce the cost of critic training in large-scale LLM alignment. The objective is as follows:

$$J_{\text{GRPO}}(\theta) = \mathbb{E}_{(q,a)\sim\mathcal{D},\,\{o^i\}_{i=1}^{G}\sim\pi_{\theta_{\text{old}}}(\cdot|q)}\Bigg[\frac{1}{\sum_{i=1}^{G}|o^i|}\sum_{i=1}^{G}\sum_{t=1}^{|o^i|}$$
$$\min\Big(r_t^i(\theta)\,\hat{A}_t^i,\, \text{clip}\big(r_t^i(\theta),\,1-\epsilon,\,1+\epsilon\big)\,\hat{A}_t^i\Big) - \beta\,\mathbb{D}_{\text{KL}}(\pi\|\pi_{\text{ref}})\Bigg],$$



(a) High-entropy tokens (b) Low-entropy tokens

Fig. 2: Word cloud visualization of token-level entropy in RTL generation: (a) high-entropy tokens and (b) low-entropy tokens.

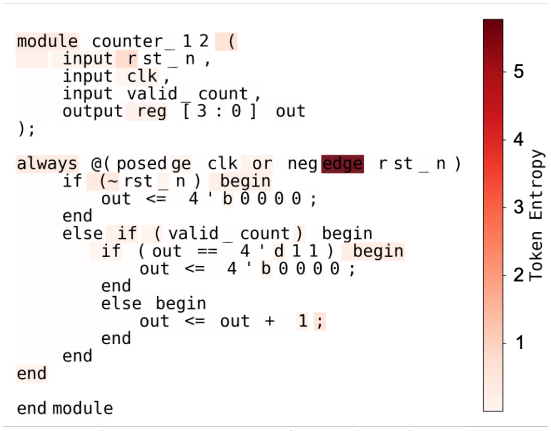


Fig. 3: Heatmap of a code snippet.

where, instead of learning a separate value function, GRPO computes token-level advantages $\hat{A}_t^i$ by normalizing rewards R^i within a group of G sampled responses $\mathbf{o} = (o^1, \ldots, o^G)$ and $r_t^i(\theta)$ for each o^i:

$$\hat{A}_t^i = \frac{r^i - \text{mean}\big(\{R^i\}_{i=1}^G\big)}{\text{std}\big(\{R^i\}_{i=1}^G\big)},$$

$$\text{where} \quad R^i = \begin{cases} 1.0, & \text{if is\_equivalent}\big(a,\, o_t^i\big), \\ 0.0, & \text{otherwise.} \end{cases}$$

Dynamic Sampling Policy Optimization (DAPO). DAPO [27] extends GRPO via clip-higher, dynamic sampling, token-Level Policy Gradient Loss, and overlong reward shaping techiniques.

Entropy-aware RLVR. Recent studies have highlighted the importance of token-level entropy in reinforcement learning with verifiable rewards. Recent works in math reasoning tasks [10]–[12] have shown that high-entropy minority tokens act as critical decision points that determine downstream reasoning and structural correctness. In contrast, low-entropy tokens typically correspond to deterministic elements such as syntax completion or port declarations, contributing little to downstream reasoning outcome. These findings suggest that RL methods should focus gradient updates on high-entropy tokens, rather than uniformly updating all tokens. Inspired by these findings, we examine entropy patterns in RTL generation and design EARL to exploit high-entropy tokens in Verilog code.

III. Token-Level Entropy in RTL Code Generation

While prior studies have examined entropy in chain-of-thought reasoning [10]–[12], we analyze entropy in RTL code generation at token level. Specifically, the entropy of the t-th token H_t is defined as:

$$H_t = -\sum_{v=1}^{V} p_{t,v} \log p_{t,v}, \tag{1}$$

$$\text{where} \quad \mathbf{p}_t = (p_{t,1}, ..., p_{t,V}) = \text{Softmax}\left(\frac{\mathbf{z}^t}{T}\right).$$

Here V is the vocabulary size, $\mathbf{z}^t \in \mathbb{R}^V$ is the pre-softmax logits, and T is the decoding temperature. Note that H_t measures the uncertainty of the predictive distribution at the t-th token.

We use a DeepSeek-7B coder model fine-tuned on our curated dataset, generating over 2.1×10^5 response tokens at decoding temperature $T = 1.0$. For each token t, entropy H_t is computed according to Eq. (1). Figure 1 presents the overall entropy distribution. The histogram is highly skewed: over 80% of tokens fall below 0.15, while only a small fraction exceeds 0.6, forming a heavy tail of high-uncertainty tokens. To identify what these extremes correspond to, Figure 2 shows the top-100 lowest-entropy tokens and top-100 highest-entropy tokens with frequency ≥ 10. Furthermore, Figure 3 overlays entropy values onto an RTL code example, highlighting uncertainty concentrations at specific structural positions.

Two consistent patterns emerge:

(i) *Deterministic bulk vs. uncertain critical tokens.* Low-entropy tokens are dominated by predictable syntax or declarative elements (e.g., `end`, `input`, `output`, whitespace), reflecting Verilog grammar. High-entropy tokens cluster around decision-making constructs with examples as follows:

- **Process/sensitivity:** `always`, `posedge`, `negedge`
- **Control flow:** `if`, `case`, ternary `?:`
- **Binding/connection:** `assign`, bus indices `[i]`
- **Module heads:** `module`, often followed by identifiers

(ii) *Entropy concentrates at structural junctions.* As shown in Fig. 3, the heatmap shows that high-entropy regions appear at sensitivity lists (`posedge`, `negedge`) and conditional branches, precisely where module behavior and timing diverge. This parallels the notion of "forking tokens" [10], but here the semantic forks represent hardware control flow and signal behavior, not natural language discourse.

These observations confirm that entropy naturally separates deterministic syntax from functional decision points in RTL code. This motivates selective updates instead of uniform updates across the entire sequence.

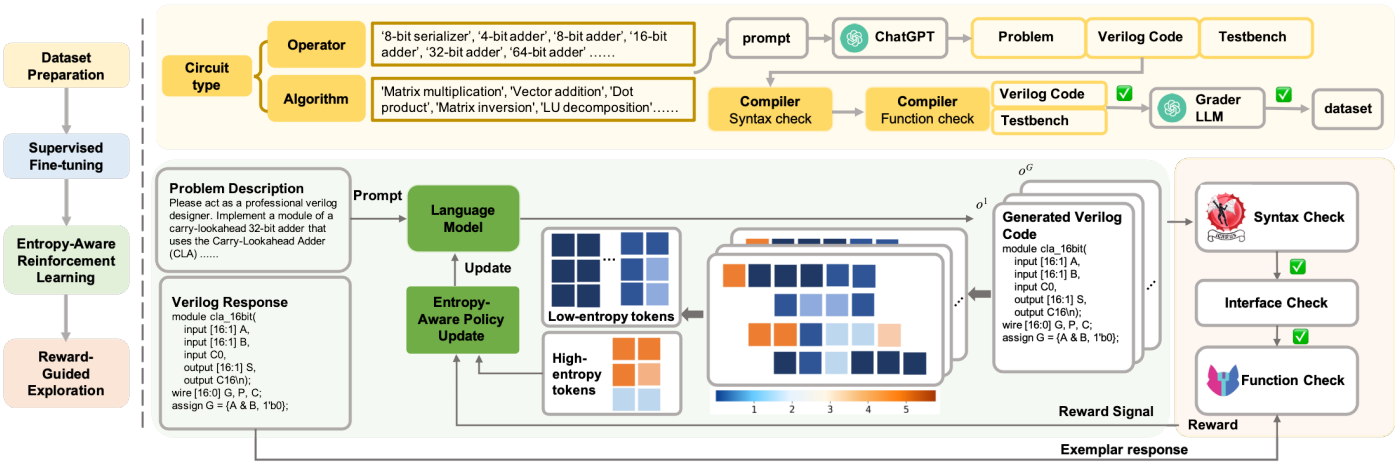


Fig. 4: EARL: an Entropy-Aware Reinforcement Learning framework.

IV. Entropy-Aware RL Framework

A. Overview of EARL

To bridge the gap between natural language specifications and functionally correct Verilog, we propose **EARL**, an *Entropy-Aware Reinforcement Learning* framework tailored for RTL generation. Figure 4 illustrates the complete pipeline. EARL addresses key limitations in prior LLM-for-Verilog approaches, including the inability of supervised fine-tuning alone to capture functional correctness signals, and the inefficiency of uniform reinforcement learning updates across long, structured RTL sequences.

The pipeline consists of four tightly coupled stages. **(1) Dataset Preparation.** We curate a synthetic dataset of natural-language specifications, Verilog implementations, and corresponding testbenches. This dataset provides functionally verified training triples that ensure syntactic validity and semantic coverage across diverse design tasks. **(2) Supervised Fine-tuning.** A pre-trained code LLM is fine-tuned on the curated dataset to establish strong syntactic fluency and structural priors. This initialization equips the model with the ability to reliably generate RTL modules with correct grammar and interface formats. **(3) Entropy-Aware Reinforcement Learning.** Building on SFT initialization, we introduce EARL, which integrates verifiable compiler/testbench signals with entropy-guided selective updates. Unlike existing RLVR methods that propagate gradients uniformly across tokens, EARL emphasizes high-entropy tokens that disproportionately govern module structure, control flow, and timing behavior, while preserving conservative updates on deterministic syntax tokens. **(4) Reward-Guided Exploration.** To align generation with downstream hardware requirements, we design a cascaded multi-signal reward incorporating syntax validity, interface consistency, and functional equivalence checking. The hierarchical reward mirrors industrial verification pipelines, ensuring that the model's exploration budget is allocated to meaningful improvements rather than syntactic noise.

By integrating dataset curation, supervised initialization, and entropy-aware RL with verifiable signals, EARL achieves robust one-pass RTL generation. The framework simultaneously improves functional correctness, stabilizes RL training on long sequences, and provides a general recipe for incorporating entropy-driven optimization into RLVR pipelines.

B. Entropy-Aware RL

Our entropy study in Section III shows that only a minority of high-entropy tokens drive effective learning. Standard RLVR pipelines update all tokens uniformly, which dilutes learning signals and wastes gradient budget on low-impact positions. To address this limitation, we introduce an entropy-based masking mechanism that selectively emphasizes high-entropy tokens during policy optimization. For each rollout response o^i, let H_t^i denote the entropy of token t. We adopt response-level quantiles to mitigate cross-prompt heterogeneity. We compute a response-level quantile threshold τ_ρ^i as $\text{Quantile}(H_t^i{}_{t=1}^{|o^i|}, \rho)$, where $\rho \in (0,1)$ denotes the quantile level. Specifically, tokens whose entropy exceeds τ_ρ^i receive full gradient updates, while low-entropy tokens are down-weighted to zero, preserving their stable distributions learned by SFT. Formally, EARL extends a DAPO-style objective with entropy gating and KL regularization:

$$J_{\text{EARL}}(\theta) = \mathbb{E}_{(q,a)\sim\mathcal{D},\{o^i\}_{i=1}^{G}\sim\pi_{\text{old}}(\cdot|q)} \Bigg[\frac{1}{\sum_{i=1}^{G}|o^i|} \sum_{i=1}^{G}\sum_{t=1}^{|o^i|} \mathbb{I}[H_t^i \geq \tau_\rho^i] \Big(\min\left(r_t^i(\theta)\hat{A}_t^i, \text{clip}\left(r_t^i(\theta), 1-\varepsilon_{low}, 1+\varepsilon_{high}\right)\hat{A}_t^i\right) - \beta\, \mathbb{D}_{\text{KL}}(\pi\|\pi_{\text{ref}}) \Big) \Bigg]$$

$$\text{s.t.} \quad 0 < \Big| \{ i \in \{1,\ldots,G\} \mid \texttt{is\_equivalent}(o^i, a) \} \Big| < G.$$

The indicator $\mathbb{I}[H_t^i \geq \tau_\rho^i]$ is the entropy mask whih differentiates the optimization strength across token type. The constraint enforces mixed-quality groups (both passing and failing samples) so that group-normalized advantages are meaningful and gradients do not degenerate.

Note that this design is optimizer-agnostic. EARL can be applied to policy gradient objectives without modifying their core

TABLE I: COMPARATIVE ANALYSIS OF VERILOG CODE GENERATION PERFORMANCE. **BOLD** INDICATES BEST RESULT. UNDERLINE INDICATES THE SECOND BEST RESULT.

Category	Method	Params.	VerilogEval-Machine		VerilogEval-Human		RTLLM	
			pass@1	pass@5	pass@1	pass@5	Syn. pass@5	Func. pass@5
Base Model	Qwen2.5-Coder	1.5B	25.6	40.8	8.3	17.9	75.1	19.2
	Qwen2.5-Coder	3B	48.4	58.9	21.3	32.7	82.9	27.3
	Qwen2.5-Coder	7B	52.7	69.7	23.9	41.1	86.2	31.0
	DeepSeek-Coder	6.7B	8.8	34.3	4.9	19.3	89.7	41.2
	CodeLlama	7B	26.1	49.1	18.8	28.6	15.2	9.3
	CodeQwen-7B-Chat	7B	29.1	61.9	14.8	36.8	16.1	11.0
Supervised Fine Tuning	VerilogEval [28]	16B	46.2	67.3	28.8	45.9	N/A	N/A
	ChipNeMo [5]	70B	53.8	N/A	27.6	N/A	28.0	20.7
	RTLLLM [29]	13B	65.3	77.2	43.7	51.8	52.6	40.7
	RTLCoder [8]	6.7B	37.2	64.9	16.9	35.7	89.1	51.7
	OriGen [7]	7B	43.1	55.1	30.8	31.9	96.5	51.7
	VeriGen [13]	16B	44.0	52.6	30.3	43.9	N/A	N/A
	HaVen [18]	6.7B	66.1	81.1	43.1	55.1	81.4	38.6
	BetterV [17]	7B	68.1	79.4	46.1	53.7	N/A	N/A
	AutoVCoder [14]	7B	68.7	79.9	<u>48.5</u>	55.9	<u>100</u>	51.7
Reinforcement Learning	VeriReason [24]	7B	<u>69.8</u>	<u>83.1</u>	47.9	<u>58.4</u>	N/A	N/A
	VeriSeek [23]	6.7B	61.6	76.9	N/A	N/A	94.8	<u>54.2</u>
EARL (ours)		7B	**72.9**	**83.9**	**49.6**	**60.2**	**100**	**68.9**

update rules. In practice, we instantiate EARL with both PPO and DAPO. By concentrating gradient updates on uncertain yet functionally critical tokens, EARL achieves sharper credit assignment, reduces wasted updates on deterministic syntax, and improves the stability of RLVR training for long, structured RTL sequences.

C. Reward Design

A central component of EARL is the design of verifiable rewards that align generation with downstream RTL correctness. Inspired by industrial EDA flows, we construct a cascaded reward hierarchy that mirrors the standard compilation–simulation–verification pipeline.

Given a natural language specification q, the model generates a candidate Verilog program o. To obtain the reward of the program, we evaluate it through three stages: **(1) Syntax Validity.** The candidate is compiled using `iverilog`. Programs that fail syntax checks are immediately assigned zero reward, preventing wasted computation on invalid code. **(2) Interface Consistency.** If syntax passes, we check whether the module name and port declarations match the specification. Partial matches receive intermediate reward, while fully aligned interfaces receive higher credit. **(3) Functional Equivalence.** Finally, we invoke the `eqy` engine from Yosys to verify functional equivalence against a reference implementation. Passing candidates receive maximal reward, while near-misses are graded slightly lower to encourage exploration without reward hacking.

D. Dataset Preparation

Training EARL requires functionally verified data to bootstrap supervised initialization and support reward evaluation during RL. To this end, we construct a synthetic dataset of specification–code–testbench triplets. The dataset generation pipeline follows two steps. **Synthetic Generation.** We employ GPT-4 to generate diverse prompts covering arithmetic operators, sequential logic, control modules, and algorithmic circuits. Each prompt is paired with a Verilog implementation and an executable testbench. Inspired by prior work [14], [17], we diversify prompts along multiple axes, including difficulty level, circuit type, and specification style. **Compiler-Driven Filtering.** To guarantee correctness, we apply a three-stage validation pipeline to all generated samples. First, syntax validity is verified using `iverilog`. Second, functional correctness is checked by executing the paired testbench. Finally, we employ a grading LLM to provide an additional layer of semantic evaluation on functional behavior. Only samples that pass all three stages are retained in the dataset.

V. EXPERIMENTS

A. Experiment setup

Models and Training. We adopt `DeepSeek-Coder-7B` as the base model. The SFT stage is conducted on our curated dataset for 3 epochs using a cosine learning rate schedule with a peak learning rate of 5×10^{-5} and 15 warm-up steps. For reinforcement learning, we apply the proposed EARL algorithm instantiated with DAPO, using group sampling ($G = 6$ rollouts per prompt) and a learning rate of 1×10^{-6}. All experiments are conducted on 4 NVIDIA A100 80GB GPUs with a global batch size of 128 and temperature 1.0 during inference.

Benchmarks. We evaluate on three standard RTL code generation benchmarks. **VerilogEval v1** [28] includes 143 machine-generated problems (`VerilogEval-Machine`) and 156 expert-written tasks (`VerilogEval-Human`), designed to test structural and semantic generation capabilities. **RTLLM**

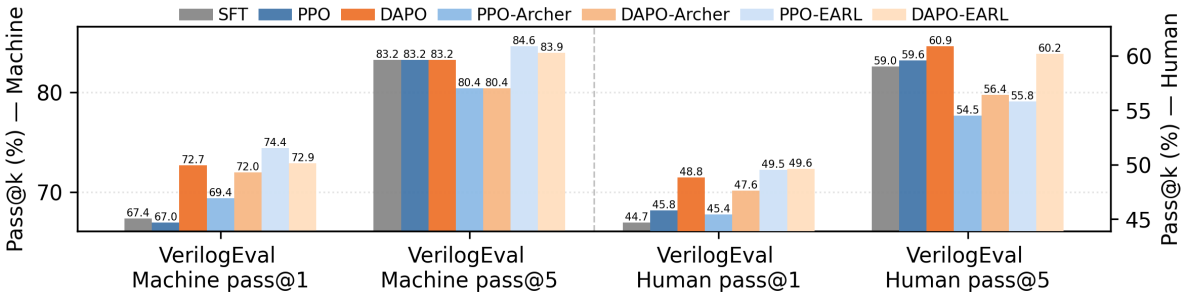


Fig. 5: Ablation study on RL algorithms and entropy mechanisms.

v1.1 [29] contains 29 diverse RTL design prompts focused on real-world functionality.

Metrics. Following prior works [5], [29], we use the pass@k metric to evaluate the probability of generating a correct solution within k attempts, where $k \in \{1, 5\}$. For each prompt, we generate $n = 5$ completions and compute:

$$\text{pass@k} := \mathbb{E}\left[1 - \frac{\binom{n-c}{k}}{\binom{n}{k}}\right],$$

where c is the number of successful completions.

B. Comparison with Competing Methods

We evaluate the syntactic and functional correctness of EARL against a wide range of baseline and state-of-the-art methods, including supervised fine-tuning (SFT) approaches and reinforcement learning with verifiable rewards (RLVR). Note that for completeness, we also report results of prompt-engineering based methods, e.g., Origen, HaVen. Their pass-k metrics are obtained from our own evaluation under the same experimental setup. Results in Table I demonstrate that EARL achieves the best performance across all benchmarks and metrics. On VerilogEval-Machine, EARL attains 72.9% pass@1 and 83.9% pass@5, surpassing the strongest RL baseline VeriReason by 3.1% and 0.8%, respectively, and outperforming the best SFT model by 4.2% in pass@1. On VerilogEval-Human, which contains expert-written tasks more aligned with real-world HDL practices, EARL achieves 49.6% pass@1 and 60.2% pass@5, both the highest among all methods. On RTLLM v1.1, EARL obtains perfect syntax pass@5 (100%) and the highest functional correctness (68.9%), outperforming the best RL method by 14.7% and the best SFT method by 17.2%. These consistent improvements demonstrate that EARL establishes a new state-of-the-art in Verilog code generation, achieving reliable one-pass correctness while remaining parameter- and data-efficient.

C. Ablation Study

1) Effect of RL Algorithms and Entropy Mechanisms: To better understand the effect of our entropy-aware reinforcement learning framework, we conduct an ablation study comparing different RL algorithms (PPO and DAPO) and different entropy mechanisms (Archer [12] and the proposed EARL). The results are summarized in Fig. 5.

We first compare PPO and DAPO without entropy masking. DAPO consistently outperforms PPO, confirming the benefits of group-based dynamic sampling in Verilog code generation. Next, we introduce entropy-aware variants. Archer applies entropy weighting, while our EARL design incorporates an entropy mask to selectively emphasize high-entropy tokens. Both methods improve over plain PPO/DAPO, but EARL achieves larger gains, particularly on VerilogEval-Machine pass@1 and VerilogEval-Human pass@5. Overall, DAPO-EARL achieves the best performance across all benchmarks, with improvements of +5.2% in pass@1 over the SFT baseline.

TABLE II: COMPARATIVE ANALYSIS OF THRESHOLD ON VERILOG CODE GENERATION PERFORMANCE. **BOLD** INDICATES BEST RESULT. UNDERLINE INDICATES THE SECOND BEST RESULT.

Quantile	VerilogEval-Machine		VerilogEval-Human		RTLLM	
	pass@1	pass@5	pass@1	pass@5	Syn.	Func.
0.0	<u>72.7</u>	<u>83.2</u>	<u>48.8</u>	**60.9**	94.8	62.0
0.2	70.5	79.7	47.3	58.4	100	**72.4**
0.4	69.4	80.4	47.6	56.4	96.1	65.5
0.6	67.8	77.6	42.3	53.2	95.1	65.5
0.8	**72.9**	**83.9**	**49.6**	<u>60.2</u>	**100**	<u>68.9</u>
0.9	71.0	79.7	45.6	56.4	96.4	65.5

2) Entropy Quantile: We further ablate the effect of the entropy quantile ρ used to construct the response-level mask. Results in Table II show that performance peaks around $\rho = 0.8$. Lower thresholds (e.g., 0.2) admit excessive noise, while higher thresholds (e.g., 1.0) exclude too many decision-critical tokens. This validates that selectively emphasizing a minority of high-entropy tokens yields the best trade-off between exploration and stability.

VI. CONCLUSION

In this work, we addressed the challenge of reliable RTL generation with large language models. Motivated by our entropy analysis, we proposed EARL, an entropy-aware reinforcement learning framework that integrates supervised initialization, verifiable reward signals, and selective policy updates. EARL emphasizes high-uncertainty tokens that govern module structure, control flow, and signal connections, while preserving stable distributions on deterministic syntax tokens. Extensive evaluation on standard benchmarks demonstrates that EARL consistently outperforms existing state-of-the-art methods.

REFERENCES

[1] J. Liu, K. Wang, Y. Chen, X. Peng, Z. Chen, L. Zhang, and Y. Lou, "Large language model-based agents for software engineering: A survey," *arXiv:2409.02977*, 2024.

[2] J. Jiang, F. Wang, J. Shen, S. Kim, and S. Kim, "A survey on large language models for code generation," *ACM Trans. Softw. Eng. Methodol.*, Jul. 2025.

[3] J. Pan, G. Zhou, C.-C. Chang, I. Jacobson, J. Hu, and Y. Chen, "A survey of research in large language models for electronic design automation," *ACM Trans. Des. Autom. Electron. Syst.*, vol. 30, no. 3, Feb. 2025.

[4] S. Thakur, B. Ahmad, Z. Fan, H. Pearce, B. Tan, R. Karri, B. Dolan-Gavitt, and S. Garg, "Benchmarking large language models for automated verilog rtl code generation," *arXiv:2212.11140*, 2022.

[5] M. Liu, T.-D. Ene, R. Kirby, C. Cheng, N. Pinckney, R. Liang, J. Alben, H. Anand, S. Banerjee, I. Bayraktaroglu, B. Bhaskaran, B. Catanzaro, A. Chaudhuri, S. Clay, B. Dally, L. Dang, P. Deshpande, S. Dhodhi, S. Halepete, E. Hill, J. Hu, S. Jain, A. Jindal, B. Khailany, G. Kokai, K. Kunal, X. Li, C. Lind, H. Liu, S. Oberman, S. Omar, G. Pasandi, S. Pratty, J. Raiman, A. Sarkar, Z. Shao, H. Sun, P. P. Suthar, V. Tej, W. Turner, K. Xu, and H. Ren, "Chipnemo: Domain-adapted llms for chip design," *arXiv:2311.00176*, 2024.

[6] K. Chang, Y. Wang, H. Ren, M. Wang, S. Liang, Y. Han, H. Li, and X. Li, "Chipgpt: How far are we from natural language hardware design," *arXiv:2305.14019*, 2023.

[7] F. Cui, C. Yin, K. Zhou, Y. Xiao, G. Sun, Q. Xu, Q. Guo, Y. Liang, X. Zhang, D. Song, and D. Lin, "Origen: Enhancing rtl code generation with code-to-code augmentation and self-reflection," in *Proceedings of the 43rd IEEE/ACM International Conference on Computer-Aided Design*, 2025.

[8] S. Liu, W. Fang, Y. Lu, J. Wang, Q. Zhang, H. Zhang, and Z. Xie, "Rtlcoder: Fully open-source and efficient llm-assisted rtl code generation technique," *Trans. Comp.-Aided Des. Integ. Cir. Sys.*, vol. 44, no. 4, p. 1448–1461, Oct. 2024.

[9] N. Wang, B. Yao, J. Zhou, Y. Hu, X. Wang, Z. Jiang, and N. Guan, "Large language model for verilog generation with code-structure-guided reinforcement learning," in *2025 IEEE International Conference on LLM-Aided Design (ICLAD)*, 2025.

[10] S. Wang, L. Yu, C. Gao, C. Zheng, S. Liu, R. Lu, K. Dang, X. Chen, J. Yang, Z. Zhang *et al.*, "Beyond the 80/20 rule: High-entropy minority tokens drive effective reinforcement learning for llm reasoning," *arXiv:2506.0193*, 2025.

[11] J. Deng, J. Chen, Z. Chen, W. X. Zhao, and J.-R. Wen, "Decomposing the entropy-performance exchange: The missing keys to unlocking effective reinforcement learning," *arXiv:2508.02260*, 2025.

[12] J. Wang, R. Liu, F. Zhang, X. Li, and G. Zhou, "Stabilizing knowledge, promoting reasoning: Dual-token constraints for rlvr," *arXiv:2507.15778*, 2025.

[13] S. Thakur, B. Ahmad, H. Pearce, B. Tan, B. Dolan-Gavitt, R. Karri, and S. Garg, "Verigen: A large language model for verilog code generation," *ACM Trans. Des. Autom. Electron. Syst.*, vol. 29, no. 3, Apr. 2024.

[14] M. Gao, J. Zhao, Z. Lin, W. Ding, X. Hou, Y. Feng, C. Li, and M. Guo, "Autovcoder: A systematic framework for automated verilog code generation using llms," in *2024 IEEE 42nd International Conference on Computer Design (ICCD)*, 2024.

[15] Y.-D. Tsai, M. Liu, and H. Ren, "Rtlfixer: Automatically fixing rtl syntax errors with large language models," *arXiv2311.16543*, 2024.

[16] Z. Yan, W. Fang, M. Li, M. Li, S. Liu, Z. Xie, and H. Zhang, "Assertllm: Generating hardware verification assertions from design specifications via multi-llms," in *Proceedings of the 30th Asia and South Pacific Design Automation Conference*, 2025.

[17] Z. Pei, H.-L. Zhen, M. Yuan, Y. Huang, and B. Yu, "Betterv: controlled verilog generation with discriminative guidance," in *Proceedings of the 41st International Conference on Machine Learning*, 2024.

[18] Y. Yang, F. Teng, P. Liu, M. Qi, C. Lv, J. Li, X. Zhang, and Z. He, "Haven: Hallucination-mitigated llm for verilog code generation aligned with hdl engineers," in *Design, Automation & Test in Europe (DATE)*, 2025.

[19] N. Stiennon, L. Ouyang, J. Wu, D. M. Ziegler, R. Lowe, C. Voss, A. Radford, D. Amodei, and P. Christiano, "Learning to summarize from human feedback," in *Proceedings of the 34th International Conference on Neural Information Processing Systems*, 2020.

[20] L. Ouyang, J. Wu, X. Jiang, D. Almeida, C. L. Wainwright, P. Mishkin, C. Zhang, S. Agarwal, K. Slama, A. Ray, J. Schulman, J. Hilton, F. Kelton, L. Miller, M. Simens, A. Askell, P. Welinder, P. Christiano, J. Leike, and R. Lowe, "Training language models to follow instructions with human feedback," in *Proceedings of the 36th International Conference on Neural Information Processing Systems*, 2022.

[21] P. Shojaee, A. Jain, S. Tipirneni, and C. K. Reddy, "Execution-based code generation using deep reinforcement learning," *Transactions on Machine Learning Research*, 2023.

[22] D. Zhang, S. Diao, X. Zou, and H. Peng, "**PLUM**: Improving code lms with execution-guided on-policy preference learning driven by synthetic test cases," 2024. [Online]. Available: https://arxiv.org/abs/2406.06887

[23] N. Wang, B. Yao, J. Zhou, X. Wang, Z. Jiang, and N. Guan, "Large language model for verilog generation with code-structure-guided reinforcement learning," *arXiv:2407.18271*, 2025.

[24] Y. Wang, G. Sun, W. Ye, G. Qu, and A. Li, "Verireason: Reinforcement learning with testbench feedback for reasoning-enhanced verilog generation," *arXiv:2505.11849*, 2025.

[25] J. Schulman, F. Wolski, P. Dhariwal, A. Radford, and O. Klimov, "Proximal policy optimization algorithms," *arXiv:1707.06347*, 2017.

[26] Z. Shao, P. Wang, Q. Zhu, R. Xu, J. Song, X. Bi, H. Zhang, M. Zhang, Y. Li, Y. Wu *et al.*, "Deepseekmath: Pushing the limits of mathematical reasoning in open language models," *arXiv:2402.03300*, 2024.

[27] Q. Yu, Z. Zhang, R. Zhu, Y. Yuan, X. Zuo, Y. Yue, W. Dai, T. Fan, G. Liu, L. Liu *et al.*, "Dapo: An open-source llm reinforcement learning system at scale," *arXiv:2503.14476*, 2025.

[28] N. Pinckney, C. Batten, M. Liu, H. Ren, and B. Khailany, "Revisiting verilogeval: A year of improvements in large-language models for hardware code generation," *ACM Trans. Des. Autom. Electron. Syst.*, Feb. 2025.

[29] Y. Lu, S. Liu, Q. Zhang, and Z. Xie, "Rtllm: An open-source benchmark for design rtl generation with large language model," in *Proceedings of the 29th Asia and South Pacific Design Automation Conference*, 2024.