Learning to Verify: Efficient Verilog Code Reranking via Collaborative Knowledge Distillation

Yiheng Shen*, Guang Yang*, Wei Zheng†, Yifan Sun‡, Fengji Zhang§, Xiang Chen¶, Fengping Xu*, Hongxing Xia*
*Nantong Normal College, Nantong 226000, China
†Northwestern Polytechnical University, Xi'an 710072, China
‡Hainan International College of Minzu University of China, Hainan 572423, China
§City University of Hong Kong, Hong Kong, China
¶Nantong University, Nantong 226000, China
Email: yiheng.s@outlook.com, novelyg@outlook.com, wzheng@nwpu.edu.cn, yunluoxingxi@gmail.com,
fengji.zhang@my.cityu.edu.hk, xchencs@ntu.edu.cn, yctcxfp@163.com, hongxing.xia@ntu.edu.cn

***Abstract*—LLMs have shown strong performance in software code generation, yet struggle with Verilog due to limited hardware domain knowledge. While sampling techniques improve pass@k by generating multiple candidates, engineers need a single reliable solution rather than uncertain alternatives. Current reranking methods face a capability gap: execution-based approaches (e.g., CodeT) rely on test case quality and incur execution overhead, while code generation models lack the discriminative capability needed for reliable correctness judgment. This paper investigates whether verification reasoning can be distilled from large expert models into a lightweight discriminator. We propose VCD-RNK, a discriminator model for Verilog code reranking built on three key insights: (1) the capability to judge functional correctness is learnable and distillable; (2) single-teacher distillation suffers from knowledge bias, which collaborative dual-teacher distillation can mitigate through complementary expertise; (3) effective supervision requires explicit reasoning traces explaining *why* code succeeds or fails, not just binary labels. After collaborative knowledge distillation, we construct VerilogJudge-47K, a dataset with expert reasoning traces, and fine-tune a 4B-parameter discriminator that approximates simulator-level verification without execution overhead. Experiments show that VCD-RNK improves pass@1 by 10.4-25.8% across multiple LLMs, achieving 93.5% of the theoretical oracle performance while without execution overhead.**

***Index Terms*—Verilog Generation, Code Rerank, Knowledge Distillation**

I. Introduction

Modern integrated circuit design complexity necessitates automated Hardware Description Language (HDL) generation (e.g., Verilog, VHDL, and SystemVerilog) to reduce costs and engineering workload [1], [2]. While LLMs have shown promising capabilities in software code generation [3], they exhibit limited performance in Verilog generation due to insufficient domain knowledge [4], [5], and retraining multiple models remains prohibitively resource-intensive [6], [7].

Current methods [8], [9] address this limitation through sampling techniques, generating multiple candidates and evaluating with pass@k metrics. As shown in Figure 1, correct implementations often exist within larger candidate pools(i.e., pass@k significantly exceeds pass@1 across diverse LLMs). However, pass@k quantifies model potential without providing a mechanism to identify the optimal implementation. In practice, engineers require a single reliable solution rather than multiple uncertain candidates [10].

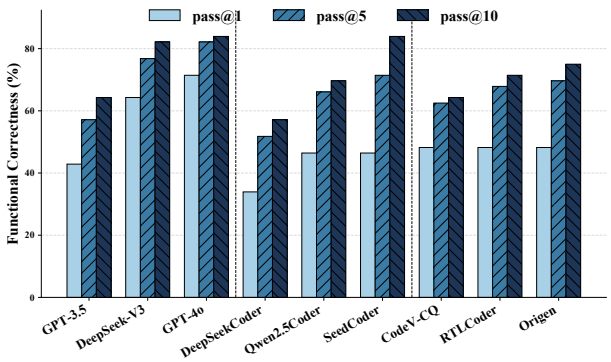


Fig. 1: Performance on ResBench benchmark.

Existing reranking approaches exhibit distinct limitations. Execution-based methods such as CodeT [10] generate test cases and verify candidates through runtime execution, but their effectiveness depends heavily on test case quality and incurs non-trivial compilation overhead. Alternatively, directly employing code generation models for correctness judgment proves unreliable, as these models are trained to *generate* code, not to *discriminate* its functional correctness, resulting in poor accuracy when applied to Verilog reranking.

Recent work has shown that functional correctness verification can be distilled from large reasoning models into lightweight discriminators. Yang et al. [11] demonstrate that a distilled 7B model can match GPT-4o's performance in Python code correctness evaluation. This establishes our first insight: *the capability to judge functional correctness is learnable and distillable*. We therefore ask whether this capability can transfer to the Verilog domain, where code exhibits inherent concurrency, timing-dependent behavior, and hardware-specific semantics that differ fundamentally from sequential

Guang Yang and Wei Zheng are the corresponding authors.

software languages. We hypothesize that successful transfer requires domain-specific distillation strategies, leading to the following research questions:

- **RQ1 (Domain Transferability):** Can functional correctness verification be effectively distilled to the Verilog domain despite its fundamentally different semantics?
- **RQ2 (Knowledge Complementarity):** Can collaborative dual-teacher distillation mitigate the systematic bias inherent in single-teacher approaches?
- **RQ3 (Supervision Quality):** Does reasoning-based supervision outperform direct model judgment for training effective discriminators?

To address these questions, we propose VCD-RNK, a lightweight discriminator for Verilog code reranking through compiler-guided dual-teacher distillation. To ensure reliable supervision (**RQ1**), we execute gold testbenches on Icarus Verilog to obtain ground-truth correctness labels rather than relying on model predictions. To capture verification expertise (**RQ2**), we distill explicit reasoning traces across three dimensions (e.g., code semantic analysis, test case generation, and functional correctness assessment) rather than binary labels alone. To mitigate single-teacher bias (**RQ3**), we employ complementary teachers (doubao-seed-1.6 and DeepSeek-R1-671B), retaining samples where at least one teacher aligns with simulator verdicts, yielding **VerilogJudge-47K** with broader error coverage. We fine-tune a 4B-parameter Qwen3 model using LoRA, enabling VCD-RNK to replicate expert verification reasoning without runtime execution overhead. Experiments show VCD-RNK improves pass@1 by 10.4-25.8% across multiple LLMs, achieving 93.5% of theoretical upper-bound performance.

Our contributions are:

- We formulate Verilog code reranking as a learnable semantic alignment problem between specifications and implementations.
- We construct **VerilogJudge-47K** through compiler-guided dual-teacher distillation and develop VCD-RNK, a 4B discriminator tailored for Verilog verification.
- We demonstrate that (1) verification reasoning transfers effectively to distilled models; (2) dual-teacher distillation mitigates single-teacher bias; (3) VCD-RNK improves pass@1 by 10.4-25.8% across diverse LLMs.

II. Background and Related Work

A. Code Generation for Verilog

Verilog features concurrency, timing-dependent behavior, and hardware-specific syntax that create unique challenges for code generation systems. Given dataset $\mathcal{D} = \{(x_i, y_i)\}_{i=1}^{|\mathcal{D}|}$ where x_i is natural language specification and y_i is Verilog implementation, a code generation model computes $p_\theta(y|x) = \prod_{t=1}^{|y|} p_\theta(y_t|x, y_{<t})$.

LLMs show promise in Verilog generation despite limited domain-specific training data. Recent work addresses this through curated datasets and supervised fine-tuning [12], [13], [14], reinforcement learning with knowledge distillation [15], and alternative representations such as AST-based templates [4].

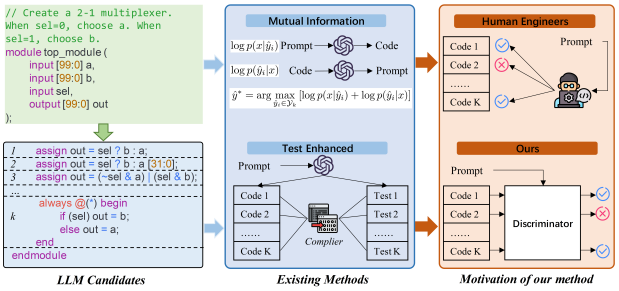


Fig. 2: Comparison of existing methods and VCD-RNK.

B. Pass@k Metric

The pass@k metric assesses whether at least one correct implementation exists among k candidates:

$$\text{pass@}k := \mathbb{E}_{x\sim\mathcal{D}}\left[1 - \frac{\binom{n-c(x)}{k}}{\binom{n}{k}}\right] \tag{1}$$

where n is total candidates, $c(x)$ is correct implementations for problem x.

C. Reranking Method for Code Generation

Reranking methods address this by scoring candidates. Given k implementations $\mathcal{Y}_k = \{\hat{y}_1, \ldots, \hat{y}_k\}$ for input x, a reranking function $R : \mathcal{X} \times \mathcal{Y} \to \mathbb{R}$ selects:

$$\hat{y}^* = \arg\max_{\hat{y}_i \in \mathcal{Y}_k} R(x, \hat{y}_i) \tag{2}$$

Existing approaches fall into two categories. **Probability-based methods** employ the probability of LLMs for candidate assessment. CodeReviewer [12] maximizes mutual information $\log p(x|\hat{y}_i) + \log p(\hat{y}_i|x)$ to select implementations with high bidirectional probability. **Test-based methods** [10], [16], [17] generate test cases alongside code and rank candidates by execution results. CodeT [10] constructs consensus sets by executing candidates against generated tests, scoring by $|\mathcal{S}_{\hat{y}}| \times |\mathcal{S}_t|$ where $\mathcal{S}_{\hat{y}}$ and $\mathcal{S}_t$ denote mutually-passing code-test pairs.

III. Method

As shown in Figure 2, VCD-RNK directly learns the semantic mapping between specifications and implementations without test case execution and complex consensus calculations.

A. Problem Formulation

Formally, given a natural language specification $x \in \mathcal{X}$ and k candidate implementations $\mathcal{Y}k = \{\hat{y}_1, \hat{y}_2, \ldots, \hat{y}_k\} \subset \mathcal{Y}$ generated by a model $p\theta(\cdot|x)$, we define a discriminator $F_\phi : \mathcal{X} \times \mathcal{Y} \to [0, 1]$ parameterized by ϕ that computes the probability of functional correctness:

$$F_\phi(x, \hat{y}i) = p(\text{correct}|x, \hat{y}_i; \phi) \tag{3}$$

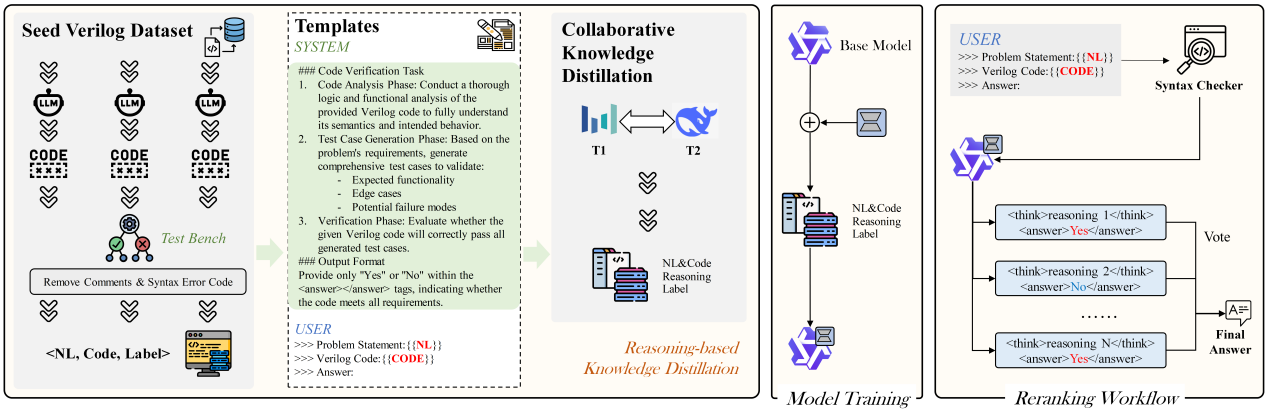


Fig. 3: Overview of our method.

Algorithm 1 Compiler-Guided Dual-Teacher Distillation

Require: Seed dataset $\mathcal{D} = \{(x_i, t_i)\}$ with specifications and testbenches; Code generator G; Teacher models T_1, T_2
Ensure: Distilled dataset $\mathcal{D}_{distill}$

// Stage 1: Compiler-Guided Label Acquisition
$\mathcal{D}_{labeled} \leftarrow \emptyset$
for each $(x, t) \in \mathcal{D}$ **do**
  $\{\hat{y}^1, \ldots, \hat{y}^k\} \leftarrow G(x)$ ▷ Sample k candidates
  for each $\hat{y}^j$ in $\{\hat{y}^1, \ldots, \hat{y}^k\}$ **do**
    $l_j \leftarrow \text{COMPILER}(\hat{y}^j, t)$ ▷ Binary correctness
    $\mathcal{D}_{labeled}$.append$(x, \hat{y}^j, l_j)$
  end for
end for
// Stage 2: Dual-Teacher Reasoning Distillation
$\mathcal{D}_{distill} \leftarrow \emptyset$
for each $(x, \hat{y}, l) \in \mathcal{D}_{labeled}$ **do**
  $r_1, p_1 \leftarrow T_1(x, \hat{y})$ ▷ Reasoning trace and prediction
  if $p_1 = l$ **then** ▷ Aligned with ground truth
    $\mathcal{D}_{distill}$.append$(x, \hat{y}, l, r_1)$
  else
    $r_2, p_2 \leftarrow T_2(x, \hat{y})$ ▷ Secondary teacher
    if $p_2 = l$ **then**
      $\mathcal{D}_{distill}$.append$(x, \hat{y}, l, r_2)$
    end if
  end if
end for
return $\mathcal{D}_{distill}$

The discriminator is trained to maximize the likelihood of correct classification:

$$\mathcal{L}\text{disc}(\phi) = -\,\mathbb{E}(x, y) \sim \mathcal{D}[c(y) \log F_\phi(x, y) + (1 - c(y)) \log(1 - F_\phi(x, y))] \quad (4)$$

During inference, the optimal implementation is selected by:

$$\hat{y}^* = \arg\max_{\hat{y}i \in \mathcal{Y}_k} F_\phi(x, \hat{y}i) \quad (5)$$

B. VCD-RNK *Design*

Figure 3 illustrates the VCD-RNK framework. VCD-RNK follows a two-stage pipeline: (1) compiler-guided labeling for reliable ground truth, and (2) collaborative dual-teacher distillation for explicit reasoning traces.

(1) Compiler-Guided Labeling To ensure reliable domain transfer (**RQ1**), we ground all supervision in objective simulation results rather than model predictions. We first use SeedCoder [18] to sample 10 candidate implementations for each specification from VeriCoder's dataset [19]. For each generated Verilog candidate, we execute the gold testbench on Icarus Verilog to obtain binary functional correctness labels, ensuring the discriminator learns from hardware-verified ground truth rather than potentially biased model judgments.

(2) Collaborative Knowledge Distillation To capture explicit reasoning traces (**RQ3**) while mitigating single-teacher bias (**RQ2**), we employ a dual-teacher distillation strategy. Algorithm 1 outlines our approach. We prompt teacher models to simulate the reasoning process of hardware engineers across three dimensions: code semantic analysis, test case generation, and functional correctness assessment. Specifically, we employ doubao-seed-1.6 as primary teacher (T_1) to generate reasoning traces for each specification-implementation pair. For instances where T_1's reasoning conclusions conflict with ground-truth labels, we invoke DeepSeek-R1-671B as secondary teacher (T_2) to provide alternative reasoning paths. Finally, we filter and merge distillation results based on compiler-verified labels, retaining only reasoning traces whose conclusions align with ground truth. This process yields **VerilogJudge-47K**, a high-quality dataset containing both correctness labels and their underlying reasoning rationale.

(3) Model Training We select Qwen3-4B as foundation model and perform knowledge injection through fine-tuning on **VerilogJudge-47K**, enabling the assimilation of Verilog-specific reasoning capabilities. To optimize computational efficiency, we implement parameter-efficient fine-tuning using LoRA [20], which freezes pre-trained weights $W_0 \in \mathbb{R}^{d \times k}$ while introducing trainable low-rank matrices $A \in \mathbb{R}^{r \times k}$ and $B \in \mathbb{R}^{d \times r}$ ($r \ll \min(d, k)$).

(4) Reranking Workflow Our workflow uses syntax checker[1] for filtering, then majority voting for final selection. We

[1]https://github.com/PyHDI/Pyverilog

Model	Pass@1	Pass@1 (Reranking k=5)							Pass@5
		Prob.	CodeReviewer	CodeRank	CodeT	Ours	*w/o SC*	*w/o MV*	
Evaluation on RTLLM-v1.1									
GPT-3.5-Turbo	32.14	-	-	32.14	**35.71**	**35.71**	*32.14*	*32.14*	39.29
DeepSeek-V3	57.14	-	-	57.14	64.29	**67.86**	*64.29*	*64.29*	67.86
GPT-4o	60.71	-	-	57.14	64.29	**67.86**	*64.29*	*64.29*	75.00
DeepSeek-Coder	32.14	28.57	21.43	28.57	28.57	**32.14**	*28.57*	*28.57*	42.86
Qwen2.5-Coder	42.86	39.29	35.71	39.29	46.43	**50.00**	*46.43*	*46.43*	53.57
Seed-Coder	42.86	39.29	50.00	46.43	42.86	**53.57**	*50.00*	*50.00*	64.29
CodeV-QC	46.43	46.43	42.86	42.86	46.43	**53.57**	*50.00*	*50.00*	60.71
RTLCoder	42.86	28.57	32.14	28.57	**42.86**	39.29	*35.71*	*35.71*	50.00
Origen	53.57	53.57	**57.14**	46.43	**57.14**	53.57	*50.00*	*50.00*	71.43
Avg.	45.64	39.29	39.88	42.06	47.62	**50.40**	*46.83*	*46.83*	58.33
Evaluation on ResBench									
GPT-3.5-Turbo	42.86	-	-	42.86	46.43	**53.57**	*50.00*	*50.00*	57.14
DeepSeek-V3	64.29	-	-	64.29	71.43	**73.21**	*69.64*	*71.43*	76.79
GPT-4o	71.43	-	-	66.07	71.43	**80.36**	*76.79*	*76.79*	82.14
DeepSeek-Coder	33.93	42.86	25.00	26.79	46.43	**50.00**	*46.43*	*46.43*	51.79
Qwen2.5-Coder	46.43	39.29	53.57	39.29	46.43	**60.71**	*55.36*	*57.14*	66.07
Seed-Coder	46.43	44.64	41.07	46.43	53.57	**64.29**	*58.93*	*60.71*	71.43
CodeV-QC	48.21	50.00	46.43	48.21	50.00	**57.14**	*53.57*	*53.57*	62.50
RTLCoder	48.21	46.43	35.71	37.50	51.79	**64.29**	*58.93*	*60.71*	67.86
Origen	48.21	41.07	44.64	48.21	50.00	**62.50**	*57.14*	*58.93*	69.64
Avg.	50.00	44.05	41.07	46.63	54.12	**62.90**	*58.53*	*59.52*	67.26

TABLE I: Performance comparison. w/o SC: without Syntax Checker, w/o MV: without Majority Voting.

perform multiple inference passes and calculate:

$$F_\phi(x, \hat{y}_i) = \frac{\sum_{j=1}^{m} \mathbb{I}[F_\phi^{(j)}(x, \hat{y}_i) = 1]}{m} \tag{6}$$

where $F_\phi^{(j)}(x, \hat{y}_i)$ represents the prediction from the j-th inference pass.

IV. Experiments

A. Setup

Datasets Our experiments evaluate reranking performance on two Verilog generation benchmarks: **RTLLM-v1.1** [21] and **ResBench** [22], which represent diverse real-world applications, including combinational logic, AI accelerators, and financial computing modules.

Evaluation Metrics We follow standard practice in code generation literature by using $\text{pass@}k$, and $\text{pass@}1$ after applying different reranking strategies to the top-k candidates.

Studied Models We evaluate across general-purpose (GPT-3.5-Turbo, GPT-4o, DeepSeek-V3), code-specialized (DeepSeek-Coder, Qwen2.5-Coder, Seed-Coder), and Verilog-specialized (CodeV-QC [23], RTLCoder [9], Origen [8]) LLMs.

Baselines We compare against Probability, CodeReviewer, CodeRank, and CodeT methods. We use CodeRankEmbed[2] for CodeRank and generate five test cases per implementation for CodeT.

[2]https://github.com/cornstack/CodeRankEmbed

B. Implementation Details

We train for 3 epochs with learning rate 1e-4 (8 hours total). During inference, majority voting aggregates 3 independent sampling passes per candidate. To prevent data leakage, we remove training samples with ROUGE-L similarity > 0.95 to evaluation benchmarks.

C. RQ1: Domain Transferability

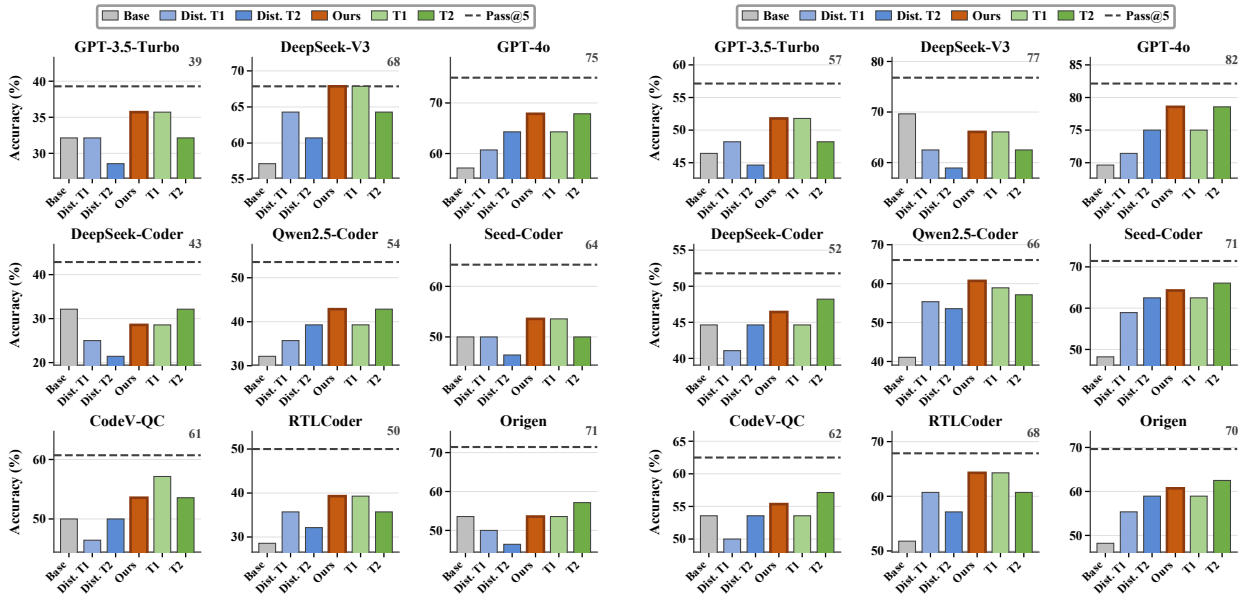
The main results are presented in Table I. For closed-source models, probability information is unavailable, resulting in "-" for Probability and CodeReviewer methods.

(1) Comparison with Baselines VCD-RNK achieves superior performance on RTLLM-v1.1 (50.40%) and ResBench (62.90%), with improvements of +5.8-16.2% over CodeT and +10.4-25.8% over original Pass@1, demonstrating better semantic alignment capture than existing methods.

(2) Comparison with Theoretical Upper Bound VCD-RNK achieves 86.4% (50.40/58.33) and 93.5% (62.90/67.26) of theoretical upper bounds on RTLLM-v1.1 and ResBench respectively, substantially narrowing the performance gap.

(3) Statistical Significance To rigorously validate the improvements observed in $\text{pass@}1$ metrics, we conducted Wilcoxon signed-rank tests comparing VCD-RNK against baselines, showing VCD-RNK's improvements are statistically significant ($p < 0.01$) in 14/18 model configurations.

(4) Ablation Results To evaluate individual component contributions, we conduct ablation studies on VCD-RNK's two



(a) Ablation results on RTLLM-v1.1.

(b) Ablation results on ResBench.

Fig. 4: Comprehensive ablation study results across both benchmarks.

inference-time components (Table I, italic columns). Removing **Syntax Checker** (pre-filtering invalid candidates) and **Majority Voting** (aggregating multiple inference passes) results in average performance drops of 3.57%/4.37% and 3.57%/3.38% on RTLLM-v1.1/ResBench respectively, demonstrating that these lightweight components provide complementary benefits in ensuring input quality and output robustness without additional training overhead.

D. RQ2: Knowledge Complementarity

Figure 4 compares VCD-RNK against single-teacher variants (Dist. T_1, Dist. T_2) and direct teacher inference (T_1: doubao-seed-1.6, T_2: DeepSeek-R1-671B).

(1) Teachers Exhibit Complementary Strengths. The two teachers show distinct performance patterns: on RTLLM-v1.1, T_1 outperforms T_2 on 5/9 models while T_2 excels on the remaining 4/9; on ResBench, T_1 leads on 3/9 models while T_2 dominates on 6/9. This validates that different teachers possess distinct knowledge biases.

(2) Single-Teacher Distillation Inherits Bias. Dist. T_1 and Dist. T_2 inherit their respective teachers' strengths and weaknesses, with neither consistently outperforming the other (RTLLM: 44.44% vs. 43.25%; ResBench: 55.95% vs. 56.55%), confirming systematic blind spots in each.

(3) Dual-Teacher Distillation Achieves Complementarity. VCD-RNK consistently surpasses both single-teacher variants, achieving +4.77%/+5.96% improvements over Dist. T_1/Dist. T_2 on RTLLM-v1.1 and +4.96%/+4.36% on ResBench, demonstrating effective knowledge complementarity through dual-teacher collaboration.

Model	Pass@1	Pass@1 (Reranking k=5)		Pass@5
		w/o Reas.	w/ Reas.	
Evaluation on RTLLM-v1.1				
GPT-3.5-Turbo	32.14	32.14	**35.71**	39.29
DeepSeek-V3	57.14	60.71	**67.86**	67.86
GPT-4o	60.71	60.71	**67.86**	75.00
DeepSeek-Coder	32.14	25.00	**32.14**	42.86
Qwen2.5-Coder	42.86	42.86	**50.00**	53.57
Seed-Coder	42.86	46.43	**53.57**	64.29
CodeV-QC	46.43	46.43	**53.57**	60.71
RTLCoder	42.86	35.71	**39.29**	50.00
Origen	53.57	46.43	**53.57**	71.43
Avg.	45.64	44.05	**50.40**	58.33
Evaluation on ResBench				
GPT-3.5-Turbo	42.86	46.43	**53.57**	57.14
DeepSeek-V3	64.29	66.07	**73.21**	76.79
GPT-4o	71.43	73.21	**80.36**	82.14
DeepSeek-Coder	33.93	41.07	**50.00**	51.79
Qwen2.5-Coder	46.43	51.79	**60.71**	66.07
Seed-Coder	46.43	55.36	**64.29**	71.43
CodeV-QC	48.21	50.00	**57.14**	62.50
RTLCoder	48.21	55.36	**64.29**	67.86
Origen	48.21	53.57	**62.50**	69.64
Avg.	50.00	54.76	**62.90**	67.26

TABLE II: Ablation on supervision quality. w/o Reas.: binary labels; w/ Reas.: reasoning traces.

E. RQ3: Supervision Quality

To evaluate whether explicit reasoning traces improve discriminator training compared to binary correctness labels alone, we train two variants: **w/o Reas.** uses only correctness labels for supervision, while **w/ Reas.** additionally incorporates reasoning traces distilled from teacher models.

Table II shows that reasoning-based supervision (**w/ Reas.**) consistently outperforms label-only training (**w/o Reas.**), with average improvements of +6.35% on RTLLM-v1.1 and +8.14% on ResBench. Notably, the label-only variant underperforms the original Pass@1 on RTLLM-v1.1 (44.05% vs. 45.64%), indicating that binary supervision alone fails to capture robust verification capabilities.

Reasoning traces provide richer supervision by explaining *why* code is correct or incorrect, enabling the discriminator to learn causal relationships between specifications and implementations rather than superficial label associations. This is particularly critical for Verilog, where functional correctness depends on subtle hardware semantics such as timing behavior and signal propagation that cannot be captured by binary labels alone.

V. Discussion

A. Efficiency Comparison

We compare computational efficiency between VCD-RNK and CodeT across different scenarios. CodeT requires three sequential stages: code generation (1-5s), test generation (1.5-5s), and test execution (1s per iteration). In contrast, VCD-RNK achieves 1.5s inference latency per instance with voting iterations, remaining substantially lower than CodeT. Importantly, VCD-RNK eliminates the test execution phase, providing significant deployment advantages.

B. Generalization Analysis

VCD-RNK operates as a model-agnostic post-hoc reranking module compatible with any Verilog generation approach, including supervised fine-tuning [12], [13] and reinforcement learning [15]. Table I shows consistent pass@1 improvements across GPT-4o, DeepSeek-V3, and Seed-Coder.

Beyond Verilog, our framework extends to other HDLs (VHDL, SystemVerilog) by adjusting the simulation backend and seed dataset, while the core distillation methodology and discriminator architecture remain unchanged.

VI. Conclusion

We introduced VCD-RNK, a discriminative reranking method for Verilog code generation that addresses the gap between model capability and practical hardware design requirements. Future directions include extending our framework to other hardware description languages to enable more complex hardware design tasks.

References

[1] P. Flake, P. Moorby, S. Golson, A. Salz, and S. Davidmann, "Verilog hdl and its ancestors and descendants," *Proceedings of the ACM on Programming Languages*, vol. 4, no. HOPL, pp. 1–90, 2020.

[2] S. Alsaqer, S. Alajmi, I. Ahmad, and M. Alfailakawi, "The potential of llms in hardware design," *Journal of Engineering Research*, 2024.

[3] G. Yang, Y. Zhou, X. Chen, X. Zhang, T. Y. Zhuo, and T. Chen, "Chain-of-thought in neural code generation: From and for lightweight language models," *IEEE Transactions on Software Engineering*, 2024.

[4] C.-T. 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 *Proceedings of the AAAI Conference on Artificial Intelligence*, vol. 39, no. 1, 2025, pp. 300–307.

[5] Y. Liu, X. Changran, Y. Zhou, Z. Li, and Q. Xu, "Deeprtl: Bridging verilog understanding and generation with a unified representation model," in *The Thirteenth International Conference on Learning Representations*, 2025.

[6] H. Shi, Z. Xu, H. Wang, W. Qin, W. Wang, Y. Wang, Z. Wang, S. Ebrahimi, and H. Wang, "Continual learning of large language models: A comprehensive survey," *ACM Computing Surveys*, 2024.

[7] M. De Carvalho, M. Pratama, J. Zhang, C. Haoyan, and E. Yapp, "Towards cross-domain continual learning," in *2024 IEEE 40th International Conference on Data Engineering (ICDE)*. IEEE, 2024, pp. 1131–1142.

[8] F. Cui, C. Yin, K. Zhou, Y. Xiao, G. Sun, Q. Xu, Q. Guo, Y. Liang, X. Zhang, D. Song *et al.*, "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*, 2024, pp. 1–9.

[9] 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," *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, 2024.

[10] B. Chen, F. Zhang, A. Nguyen, D. Zan, Z. Lin, J.-G. Lou, and W. Chen, "Codet: Code generation with generated tests," in *The Eleventh International Conference on Learning Representations*, 2023.

[11] G. Yang, Y. Zhou, X. Chen, W. Zheng, X. Hu, X. Zhou, D. Lo, and T. Chen, "Code-diting: Automatic evaluation of code generation without references or test cases," in *2025 40th IEEE/ACM International Conference on Automated Software Engineering (ASE)*. IEEE/ACM, 2025, pp. 170–182.

[12] M. Liu, N. Pinckney, B. Khailany, and H. Ren, "Verilogeval: Evaluating large language models for verilog code generation," in *2023 IEEE/ACM International Conference on Computer Aided Design (ICCAD)*. IEEE, 2023, pp. 1–8.

[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 Transactions on Design Automation of Electronic Systems*, vol. 29, no. 3, pp. 1–31, 2024.

[14] Y. Zhang, Z. Yu, Y. Fu, C. Wan, and Y. C. Lin, "Mg-verilog: Multi-grained dataset towards enhanced llm-assisted verilog generation," in *2024 IEEE LLM Aided Design Workshop (LAD)*. IEEE, 2024, pp. 1–5.

[15] Y. Zhu, D. Huang, H. Lyu, X. Zhang, C. Li, W. Shi, Y. Wu, J. Mu, J. Wang, Y. Zhao *et al.*, "Codev-r1: Reasoning-enhanced verilog generation," *arXiv preprint arXiv:2505.24183*, 2025.

[16] Z. Sun, Y. Wan, J. Li, H. Zhang, Z. Jin, G. Li, and C. Lyu, "Sifting through the chaff: On utilizing execution feedback for ranking the generated code candidates," in *Proceedings of the 39th IEEE/ACM International Conference on Automated Software Engineering*, 2024, pp. 229–241.

[17] Z. Zhao, R. Qiu, C. Lin, G. L. Zhang, B. Li, and U. Schlichtmann, "Vrank: Enhancing verilog code generation from large language models via self-consistency," in *2025 26th International Symposium on Quality Electronic Design (ISQED)*. IEEE, 2025, pp. 1–7.

[18] Y. Zhang, J. Su, Y. Sun, C. Xi, X. Xiao, S. Zheng, A. Zhang, K. Liu, D. Zan, T. Sun *et al.*, "Seed-coder: Let the code model curate data for itself," *arXiv preprint arXiv:2506.03524*, 2025.

[19] A. Wei, H. Tan, T. Suresh, D. Mendoza, T. S. Teixeira, K. Wang, C. Trippel, and A. Aiken, "Vericoder: Enhancing llm-based rtl code generation through functional correctness validation," *arXiv preprint arXiv:2504.15659*, 2025.

[20] E. J. Hu, Y. Shen, P. Wallis, Z. Allen-Zhu, Y. Li, S. Wang, L. Wang, W. Chen *et al.*, "Lora: Low-rank adaptation of large language models." *ICLR*, vol. 1, no. 2, p. 3, 2022.

[21] Y. Lu, S. Liu, Q. Zhang, and Z. 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)*. IEEE, 2024, pp. 722–727.

[22] C. Guo and T. Zhao, "Resbench: A resource-aware benchmark for llm-generated fpga designs," in *Proceedings of the 15th International Symposium on Highly Efficient Accelerators and Reconfigurable Technologies*, 2025, pp. 25–34.

[23] Y. Zhao, D. Huang, C. Li, P. Jin, Z. Nan, T. Ma, L. Qi, Y. Pan, Z. Zhang, R. Zhang *et al.*, "Codev: Empowering llms for verilog generation through multi-level summarization," *arXiv preprint arXiv:2407.10424*, 2024.