

EstCoder: A RTL Code Generator based on Static Functional Estimation

Qi Xiong¹, Renzhi Chen³, Zhigang Fang¹, Bowei Wang¹, Yingjie Zhou¹, Libo Huang¹, Lei Wang^{3,2}

¹National University of Defense Technology, Changsha, China, ²Qiyuan Laboratory, Beijing, China

³Defense Innovation Institute, Academy of Military Sciences, Beijing, China

{xiongqi}@nudt.edu.cn, {wangleia}@qiyuanlab.com, {chenrenzhi}@qiyuanlab.com

Abstract—Optimizing register transfer level (RTL) code is of vital importance in hardware design. Large language models (LLMs) provide new methods for the automatic generation and optimization of RTL code. However, existing methods for generating RTL code often focus on model fine-tuning and the use of various expansion techniques to enhance the RTL code generation capabilities, lacking attention to the functional correctness. To address this issue, we propose EstCoder, an LLM-powered collaborative agent framework for RTL code generation based on static functional score estimation. EstCoder operates a three-stage paradigm: Generation, Estimation and Correction. During the stages, the functional estimation agent statically evaluates the generated code based on score and assessment results, and decides whether to output the code directly, return it for regeneration, or forward it to the code correction agent. This framework can be applied to various LLMs that designed for RTL code generation, further enhancing the correctness of the generated code. By providing quantitative scores and human-readable requirements comparisons, it improves the transparency of AI-assisted RTL code generation. Experiments show that EstCoder significantly improves the correctness of RTL code generation by generic LLM by 3.2%-9.0%, demonstrating the practical value of our system.

Index Terms—RTL Code Generation, Large Language Model, Functional Estimation

I. INTRODUCTION

Large language models (LLMs) have demonstrated outstanding performance in natural language processing (NLP) tasks [1], enabling a wide range of applications such as text generation [2], translation [3], and question-answering [4]. Paper [5]–[7] show that LLM can produce syntactically correct code from natural language prompts, reducing development time and improving productivity. This success has naturally extended interest from general-purpose programming to domain-specific languages. In particular, the application of LLMs to hardware description languages (HDLs), such as Verilog and VHDL, has attracted considerable attention [8]–[11].

Currently, LLM-assisted RTL code generation primarily involves three directions: constructing datasets for fine-tuning [12]–[20], applying extension techniques such as Retrieval-Augmented Generation (RAG) and Graph-Enhanced methods

This work was supported in part by the National Natural Science Foundation of China under Grants 62372461, 62032001, 62203457, and 62406335. (corresponding author: Lei Wang)

¹Lei Wang is with Defense Innovation Institute, AMS. E-mail:wangleia@qiyuanlab.com.

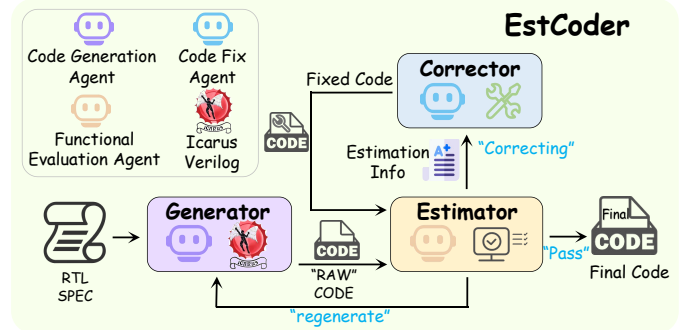


Fig. 1: The overview of EstCoder. EstCoder operates a three-stage paradigm of Generation, Estimation and Correction.

[21], [22], and exploring code generation and code correction approaches [23]–[25]. Among these, research on using LLMs to generate and fix RTL code is highly universal, as its methods can also be integrated into the first two approaches to further improve the correctness of RTL code. However, the correctness verification of RTL code relies heavily on testbenches. Manual creation of testbenches is time-consuming and labor-intensive, while automatically generated testbenches by LLMs may lack comprehensiveness and accuracy [26], [27]. Therefore, most existing methods solely focus on syntactic correctness. LLM-generated and LLM-fixed RTL code often cannot be seamlessly integrated, thus are usually studied separately [26], [28]. Furthermore, most existing code fixing studies rely on binary pass/fail evaluation, which offers no intermediate feedback and thus lacks fine-grained guidance for iterative model optimization.

In this work, we introduce EstCoder, an LLM-powered collaborative agent framework for RTL code generation based on static functional score estimation, as shown in Figure 1. Static functional score estimation enhances the transparency of AI-assisted RTL code generation by providing quantified scores, offering intermediate feedback and fine-grained guidance for code correction. By estimating code functional correctness rather than labor-intensive testbenches, it bridges code generation and correction. Moreover, our approach can be applied to various LLMs designed for RTL code generation.

II. ESTCODER METHOD

EstCoder operates a three-stage paradigm: Generation, Estimation and Correction.

The code generation part consists of a Code Generation Agent and a grammar checker. The RTL code specification is fed into the code generation agent, which produces code that is iteratively refined until it passes a syntax check. For the errors that can be directly corrected, we summarize them as an error reminder and attach it to the prompt of the Code Generation Agent, guiding it to generate RTL code.

Then, the functional estimation agent performs static functional score estimation, generating both a score and an assessment result to judge the code quality. Inspired by the software engineering field and reference [29], firstly, the generated RTL code is input to the reverse requirement generator to reconstruct requirements. A pre-trained model then scores semantic similarity and completeness against the original requirements. These scores are provided to the LLM Judger to derive a comprehension score, and together they form the final VFCS (Functional Correctness) functional score.

The VFCS score is calculated using the following formula:

$$v_s = \sigma(S_{\text{token}} + S_{\text{sentence}} + C + L_{\text{LLM}}) \quad (1)$$

where v_s represents the VFCS score. $\sigma(z) = \frac{w}{1+e^{-z}}$, where w is the coefficient of each score. S_{token} encodes the input text at the token level with the pre-trained language model roberta-large. S_{sentence} uses the pre-trained model all-MiniLM-L6-v2 in the sentence Transformers library. C extracts key nouns, verbs, and appropriate nouns from the input, and calculates penalties based on missing or additional keywords. C is calculated as the following function:

$$C = \max\left(0, 1 - \frac{\phi}{\max(T_{\text{total}}, 1)}\right). \quad (2)$$

where ϕ represents the penalty, calculated as the sum of the sizes of the missing and additional but not in keyword sets. T_{total} represents the total number of keywords in both keyword sets. L_{LLM} takes original code requirements along with the code generated by the code generator as input, and supplements them with S_{token} , S_{sentence} , and C .

If the estimation indicates that the code quality is unacceptably low, it is returned to the code generation agent for regeneration; otherwise, if the estimation suggests that the code fails but still has potential, it proceeds to the code correction agent in the third stage, where the code is corrected and re-estimated. The Code Correction Agent operates during the conversation stage based on a large language model (LLM), using the model’s reasoning ability to repair RTL code. The entire code repair process is divided into Reasoning and Correcting two stages. In the Reasoning stage, the LLMs are directed to attribute the reason for not passing the functional estimation, then the LLMs will propose a method based on natural language to solve these errors. In the Correcting stage, the LLMs are guided to modify the RTL code.

TABLE I: Main results

Method	LLM Model	VerilogEval-v1-Human (Pass@1)	VerilogEval-v2 (Pass@1)
Generic LLM	Qwen-Plus	62.2	58.3
	Qwen3-32b	45.5	51.3
	DeepSeek-Chat	73.1	65.4
	GPT-4o	50.6	62.2
RTL-Specified LLM	RTLCoder [13]	41.6	36.5
	ITERTL [32]	42.9	N/A
	CodeV [18]	53.2	N/A
LLM + RTLFixer [22]	GPT-3.5 Turbo	46.4	44.9
	GPT-4 Turbo	65.0	65.4
VeriAssist [23]	Claude-3	41.6	N/A
	GPT-3.5	34.4	N/A
	GPT-4	50.5	N/A
OriGen [15]	DeepSeek-Coder-7B + LoRA	54.4	N/A
AutoVCoder [17]	CodeQwen1.5-7B	48.5	N/A
EstCoder(Ours)	Qwen-Plus	66.7 (+4.5)	63.5 (+5.2)
	Qwen3-32b	52.6 (+7.1)	60.3 (+9.0)
	DeepSeek-Chat	76.3 (+3.2)	70.5 (+5.1)
	GPT-4o	58.3 (+7.7)	67.3 (+5.1)

Note: N/A denotes that evaluation on VerilogEval-v2 was not yet available when this paper was submitted.

III. EXPERIMENT

To verify the effectiveness of the EstCoder, We adopt two representative Verilog generation benchmarks, VerilogEval-v1-Human dataset [30] and VerilogEval-v2 dataset [31].

We provided RTL code specification to the EstCoder and used the testbench accompanying the dataset to verify the generated RTL code. We selected four distinct LLMs as baseline models: Qwen-Plus, Qwen3-32b, DeepSeek-Chat, GPT-4o and compared them with other different models and methods. We used the pass rate (the number of tasks verified by the test benchmark divided by the total number of code generation tasks) as the evaluation metric.

The results of functional correctness between our method and existing baselines are shown in Table I. Our solution in DeepSeek-Chat achieves 76.3% and 70.5% accuracy in pass@1 evaluations on VerilogEval-v1-Human and VerilogEval-v2, respectively. Based on strong vanilla models such as GPT-4o, our method achieved significant improvements of 7.7% on VerilogEval-v1-Human and 5.1% on VerilogEval-v2.

IV. CONCLUSION

In this work, we developed an innovative framework for generating RTL code based on LLM-Agent, significantly improving the performance of generic LLMs on the VerilogEval dataset. Our approach is fully open-source: <https://anonymous.4open.science/r/EstCoder-X2K7/>, allowing for further experimentation and adaptation by research community.

REFERENCES

- [1] T. B. Brown, B. Mann, N. Ryder, M. Subbiah, J. Kaplan, P. Dhariwal, A. Neelakantan, P. Shyam, G. Sastry, A. Askell, S. Agarwal, A. Herbert-Voss, G. Krueger, T. Henighan, R. Child, A. Ramesh, D. M. Ziegler, J. Wu, C. Winter, C. Hesse, M. Chen, E. Sigler, M. Litwin, S. Gray, B. Chess, J. Clark, C. Berner, S. McCandlish, A. Radford, I. Sutskever, and D. Amodei, "Language models are few-shot learners," in *Advances in Neural Information Processing Systems 33: Annual Conference on Neural Information Processing Systems*, 2020.
- [2] Y. Wu, "Large language model and text generation," in *Natural language processing in biomedicine: A practical guide*. Springer, 2024, pp. 265–297.
- [3] D. Elshin, N. Karpachev, B. Gruzdev, I. Golovanov, G. Ivanov, A. Antonov, N. Skachkov, E. Latypova, V. Layner, E. Enikeeva *et al.*, "From general llm to translation: How we dramatically improve translation quality using human evaluation data for llm finetuning," in *Proceedings of the Ninth Conference on Machine Translation*, 2024, pp. 247–252.
- [4] M. A. Ehsan, A. Hasan, K. B. Shahnoor, and S. S. Tasneem, "Automatic question & answer generation using generative large language model (llm)," *arXiv preprint arXiv:2508.19475*, 2025.
- [5] J. Yang, C. E. Jimenez, A. Wettig, K. Lieret, S. Yao, K. Narasimhan, and O. Press, "Swe-agent: Agent-computer interfaces enable automated software engineering," *Advances in Neural Information Processing Systems*, vol. 37, pp. 50 528–50 652, 2024.
- [6] S. Holt, M. R. Luyten, and M. van der Schaar, "L2MAC: large language model automatic computer for extensive code generation," in *The Twelfth International Conference on Learning Representations, (ICLR) 2024, Vienna, Austria, May 7-11, 2024*.
- [7] K. Zhang, J. Li, G. Li, X. Shi, and Z. Jin, "Codeagent: Enhancing code generation with tool-integrated agent systems for real-world repo-level coding challenges," in *Proceedings of the 62nd Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers), ACL 2024, Bangkok, Thailand, August 11-16, L. Ku, A. Martins, and V. Srikumar, Eds., 2024*, pp. 13 643–13 658.
- [8] F. Cassano, J. Gouwar, D. Nguyen, S. Nguyen, L. Phipps-Costin, D. Pinckney, M.-H. Yee, Y. Zi, C. J. Anderson, M. Q. Feldman *et al.*, "Multipl-e: A scalable and polyglot approach to benchmarking neural code generation," *IEEE Transactions on Software Engineering*, vol. 49, no. 7, pp. 3675–3691, 2023.
- [9] G. Dessouky, D. Gens, P. Haney, G. Persyn, A. Kanuparthi, H. Khattri, J. M. Fung, A.-R. Sadeghi, and J. Rajendran, "{HardFails}: insights into {software-exploitable} hardware bugs," in *28th USENIX Security Symposium (USENIX Security 19)*, 2019, pp. 213–230.
- [10] B. Ahmad, S. Thakur, B. Tan, R. Karri, and H. Pearce, "On hardware security bug code fixes by prompting large language models," *IEEE Transactions on Information Forensics and Security*, vol. 19, pp. 4043–4057, 2024.
- [11] K. Laeuffer, B. Fajardo, A. Ahuja, V. Iyer, B. Nikolić, and K. Sen, "Rtl-repair: Fast symbolic repair of hardware design code," in *Proceedings of the 29th ACM International Conference on Architectural Support for Programming Languages and Operating Systems, Volume 3*, 2024, pp. 867–881.
- [12] 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. [Online]. Available: <https://doi.org/10.1145/3643681>
- [13] S. Liu, W. Fang, Y. Lu, J. Wang, Q. Zhang, H. Zhang, and Z. Xie, "Rtl-coder: Fully open-source and efficient llm-assisted rtl code generation technique," *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, vol. 44, no. 4, pp. 1448–1461, 2025.
- [14] A. Wei, H. Tan, T. Suresh, D. Mendoza, T. S. F. X. Teixeira, K. Wang, C. Trippel, and A. Aiken, "Vericoder: Enhancing llm-based RTL code generation through functional correctness validation," *CoRR*, vol. abs/2504.15659, 2025.
- [15] 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*. New York, NY, USA: Association for Computing Machinery, 2025. [Online]. Available: <https://doi.org/10.1145/3676536.3676830>
- [16] Z. Pei, H. Zhen, M. Yuan, Y. Huang, and B. Yu, "Betterv: Controlled verilog generation with discriminative guidance," in *Forty-first International Conference on Machine Learning, ICML 2024, Vienna, Austria, July 21-27, 2024*, 2024.
- [17] 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, pp. 162–169.
- [18] Y. Zhao, D. Huang, C. Li, P. Jin, M. Song, Y. Xu, Z. Nan, M. Gao, T. Ma, L. Qi *et al.*, "Codev: Empowering llms with hdl generation through multi-level summarization," *arXiv preprint arXiv:2407.10424*, 2024.
- [19] M. Liu, Y. Tsai, W. Zhou, and H. Ren, "Craftrtl: High-quality synthetic data generation for verilog code models with correct-by-construction non-textual representations and targeted code repair," in *The Thirteenth International Conference on Learning Representations, ICLR 2025, Singapore, April 24-28, 2025*.
- [20] C. Deng, Y. Tsai, G. Liu, Z. Yu, and H. Ren, "Scalertl: Scaling llms with reasoning data and test-time compute for accurate RTL code generation," *CoRR*, 2025.
- [21] M. Akyash, K. Z. Azar, and H. M. Kamali, "RTL++: graph-enhanced LLM for RTL code generation," *CoRR*, 2025.
- [22] Y. Tsai, M. Liu, and H. Ren, "Rtlfixer: Automatically fixing RTL syntax errors with large language models," *CoRR*, 2023.
- [23] H. Huang, Z. Lin, Z. Wang, X. Chen, K. Ding, and J. Zhao, "Towards llm-powered verilog RTL assistant: Self-verification and self-correction," *CoRR*, 2024.
- [24] Y. Zhao, H. Zhang, H. Huang, Z. Yu, and J. Zhao, "MAGE: A multi-agent engine for automated RTL code generation," *CoRR*, 2024.
- [25] J. Zhang, C. Liu, and H. Li, "Understanding and mitigating errors of llm-generated rtl code," *arXiv preprint arXiv:2508.05266*, 2025.
- [26] R. Qiu, G. L. Zhang, R. Drechsler, U. Schlichtmann, and B. Li, "Autobench: Automatic testbench generation and evaluation using llms for hdl design," in *Proceedings of the 2024 ACM/IEEE International Symposium on Machine Learning for CAD*, 2024, pp. 1–10.
- [27] —, "Correctbench: Automatic testbench generation with functional self-correction using llms for hdl design," in *2025 Design, Automation & Test in Europe Conference (DATE)*. IEEE, 2025, pp. 1–7.
- [28] J. Bhandari, J. Knechtel, R. Narayanaswamy, S. Garg, and R. Karri, "Llm-aided testbench generation and bug detection for finite-state machines," *CoRR*, 2024.
- [29] A. A. N. Ponnusamy, "Bridging llm-generated code and requirements: Reverse generation technique and sbc metric for developer insights," *arXiv preprint arXiv:2502.07835*, 2025.
- [30] 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)*, 2023.
- [31] N. R. Pinckney, C. Batten, M. Liu, H. Ren, and B. Khailany, "Revisiting verilogeval: Newer llms, in-context learning, and specification-to-rtl tasks," *CoRR*, 2024.
- [32] P. Wu, N. Guo, X. Xiao, W. Li, X. Ye, and D. Fan, "Itertl: An iterative framework for fine-tuning llms for rtl code generation," in *2025 IEEE International Symposium on Circuits and Systems (ISCAS)*. IEEE, 2025, pp. 1–5.