

VANCE: The SystemVerilog Assistant That Runs on a Laptop

Patrick Ostler

Ostler AI

patrick{at}ostlerai{dot}com

Abstract– Since 2022, LLM adoption among ASIC/FPGA developers has grown, though privacy, availability, and cost remain barriers. We present the Verilog Assistant Novel Chatbot Entity (VANCE): a 4-billion parameter, open-weight model based on Qwen3. VANCE achieves competitive performance with GPT-4o in spec-to-RTL generation while remaining small enough to run on a standard engineering laptop without a discrete GPU.

I. INTRODUCTION

We introduce VANCE, a SystemVerilog-tuned LLM that outperforms many foundation models and most domain-specific models in generating RTL from natural-language specifications. By limiting the footprint to 4 billion parameters, VANCE improves privacy and reduces cost by enabling local deployment on engineering laptops. This paper details the data curation, training pipeline, and benchmark results.

II. ARCHITECTURE & METHODS

For VANCE, we chose Qwen3-4B-Base as our base model for its reasoning capabilities and permissive licensing [1]. Training data is formatted using Qwen’s default chat templates and contains virtually no LLM-generated HDL code – all HDL code samples pre-date ChatGPT’s release on Nov 30, 2022 or were created in-house.

A. Data Curation & Processing

Organic SystemVerilog examples were harvested from permissively licensed GitHub repositories (via BigQuery, from a Nov 26, 2022 checkpoint) and a 2015 snapshot of OpenCores. Samples were syntax-checked via slang [2] and filtered by AI agents for utility, clarity, practical use, and teaching value. Samples scoring below 50% were rejected. From this, ~5k code samples remain, and “specification” prompts were created by pairing interface data harvested from slang with design descriptions created by AI agents.

To supplement this, we used rule-based code generators (including portions of CraftRTL [3]) to create 33k correct-by-construction samples with prompts and testbenches. Additional data includes: 200 hand-coded spec-to-RTL prompt and code pairs, 1k boolean algebra simplification prompts with step-by-step solutions, 94k synthetic Q&A pairs generated from public SystemVerilog resources, 5k Stack Overflow/Electrical Engineering SE pairs, 10k verified Python prompt-code pairs [4], and 600 verified C++ prompt-code pairs [5]. Training data was filtered to exclude benchmark sources (e.g., HDLBits solutions on GitHub).

B. Model Training

VANCE was trained on four NVIDIA RTX 3090 GPUs using a two-stage process: Supervised Fine-Tuning (SFT) followed by Reinforcement Learning with Verifiable Rewards (RLVR).

1) *SFT*: For SFT, we used Axolotl [6] and the full dataset described above. We configured our runs for a full parameter fine-tune with a 12k token context window, a batch size of 1 with sample packing enabled, and a learn rate of $7.5e-5$ with cosine decay. The 94k synthetic SystemVerilog Q&A pairs are trained for only 2 epochs to prevent overfitting while prioritizing code generation, while the remainder of the dataset is trained for 5 epochs.

2) *RLVR*: For RLVR, we employed VerL [7] on a representative subset of 3k specification-code pairs. To fit memory constraints, we used low-rank adapter (LoRA) parameter-efficient fine-tuning (PEFT) with a rank of 32 and alpha of 32. We further reduce the memory footprint by decreasing the context window from 12k tokens (used in SFT) to 5k tokens and selected only samples where the reference code is shorter than 2k tokens. We used Group Sequence Policy Optimization (GSPO) [8] as the RL algorithm. Rewards for the generated code samples (rollouts) are binary (pass/fail), determined via Icarus Verilog [9] if a testbench is available, or via Yosys EQY [10] formal equivalence checking against the golden reference code if no testbench is present.

TABLE I
COMPARISON OF VANCE TO POPULAR FOUNDATION MODELS AND VERILOG-SPECIFIC MODELS ON THE VERILOGEVALV2 SPEC-TO-RTL BENCHMARK [11]. THE BEST RESULTS PER CATEGORY ARE HIGHLIGHTED IN BOLD.

Type	Model	Size	Pass@1 (%)	Source
Foundational	Gemini 3 Flash	-	90.7	[12]
	Gemma-4-31B-it	31B	88.0	[13]
	Kimi-K2.5	1000B	87.2	[13]
	GPT-5 mini	-	81.8	[12]
	DeepSeek R1-0528	685B	81.5	[13]
	gpt-oss-120b	120B	78.8	[13]
	gpt-oss-20b	21.5B	69.7	[13]
	Claude Sonnet 4	-	67.2	[12]
	GPT-4o	-	64.1	[14]
	Qwen3-4B-Thinking-2507	4B	45.4	This work
	Qwen3-4B-Instruct-2507	4B	33.1	This work
Verilog-tuned	ScaleRTL (Qwen3) [15]	32B	76.3	[15]
	CodeV-R1-7B (Qwen2.5-Coder) [14]	7B	68.8	[14]
	CraftRTL (Starcoder2) [3]	16B	68.0	[3]
	QiMeng-CRUX (Qwen2.5-Coder) [16]	7B	64.7	[16]
	OriGen (DeepSeek-Coder) [17]	7B	54.2	[13]
	CodeV (CodeQwen1.5) [18]	7B	48.4	[19]
	HaVen (CodeQwen1.5) [20]	7B	47.3	[13]
	RTLCoder (DeepSeek-Coder) [21]	7B	39.5	[13]
	VeriReason (Qwen2.5) [22]	7B	34.4	[12]
Ours	VANCE (Qwen3)	4B	69.4	This work

III. EXPERIMENTAL EVALUATION

We evaluated VANCE in a zero-shot setting on the VerilogEvalV2 benchmark [11], where it achieved a pass@1 rate of 69.4% and a pass@8 rate of 84.0% over $n = 8$ samples per prompt. Table I compares these results against instruction-tuned and thinking variants of Qwen3-4B, which were evaluated in-house, as well as several foundational and Verilog-specific models based on externally reported performance.

To verify the claim that VANCE will run on a laptop, we deployed 8-bit GGUF quantization of VANCE on an HP laptop (AMD Ryzen 5 5500U, 32GB RAM), achieving a speed of 6 tokens/sec using llama.cpp [23] for CPU-only inference.

IV. CONCLUSION

VANCE demonstrates that domain-specific tuning and verifiable rewards can allow a 4B parameter model to reach GPT-4o-level RTL generation. This democratizes high-quality AI assistance by enabling local execution. Future work includes extending VANCE to multi-file projects, agentic flows, and SVA/UVM generation.

REFERENCES

- [1] A. Yang, A. Li, B. Yang, B. Zhang, B. Hui, B. Zheng, B. Yu, C. Gao, C. Huang, C. Lv, C. Zheng, D. Liu, F. Zhou, F. Huang, F. Hu, H. Ge, H. Wei, H. Lin, J. Tang, J. Yang, J. Tu, J. Zhang, J. Yang, J. Yang, J. Zhou, J. Zhou, J. Lin, K. Dang, K. Bao, K. Yang, L. Yu, L. Deng, M. Li, M. Xue, M. Li, P. Zhang, P. Wang, Q. Zhu, R. Men, R. Gao, S. Liu, S. Luo, T. Li, T. Tang, W. Yin, X. Ren, X. Wang, X. Zhang, X. Ren, Y. Fan, Y. Su, Y. Zhang, Y. Zhang, Y. Wan, Y. Liu, Z. Wang, Z. Cui, Z. Zhang, Z. Zhou, and Z. Qiu, “Qwen3 technical report,” *arXiv preprint arXiv:2505.09388*, 2025.
- [2] Mike Popoloski and slang maintainers and contributors, “slang - systemverilog language services,” 2015. [Online]. Available: <https://github.com/MikePopoloski/slang>
- [3] M. Liu, Y.-D. 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,” 2024. [Online]. Available: <https://arxiv.org/abs/2409.12993>
- [4] N. M. Petit, “Vezora’s codetester dataset,” <https://huggingface.co/datasets/Vezora/Tested-143k-Python-Alpaca>, 2024.
- [5] Redhanuman, “Soltra codeforces cpp elite,” <https://huggingface.co/datasets/Redhanuman/soltra-codeforces-cpp-elite-10k>, 2025.

- [6] Axolotl maintainers and contributors, “Axolotl: Open source llm post-training,” 2023. [Online]. Available: <https://github.com/axolotl-ai-cloud/axolotl>
- [7] G. Sheng, C. Zhang, Z. Ye, X. Wu, W. Zhang, R. Zhang, Y. Peng, H. Lin, and C. Wu, “Hybridflow: A flexible and efficient rlhf framework,” *arXiv preprint arXiv: 2409.19256*, 2024.
- [8] C. Zheng, S. Liu, M. Li, X.-H. Chen, B. Yu, C. Gao, K. Dang, Y. Liu, R. Men, A. Yang, J. Zhou, and J. Lin, “Group sequence policy optimization,” 2025. [Online]. Available: <https://arxiv.org/abs/2507.18071>
- [9] S. Williams and M. Baxter, “Icarus verilog: open-source verilog more than a year later,” *Linux Journal*, vol. 2002, p. 3, 01 2002.
- [10] C. Wolf, J. Glaser, and J. Kepler, “Yosys-a free verilog synthesis suite,” 2013. [Online]. Available: <https://api.semanticscholar.org/CorpusID:202611483>
- [11] N. Pinckney, C. Batten, M. Liu, H. Ren, and B. Khailany, “Revisiting verilogval: Newer llms, in-context learning, and specification-to-rtl tasks,” 2024. [Online]. Available: <https://arxiv.org/abs/2408.11053>
- [12] L. Collini, A. Hennesee, P. Yubeaton, S. Garg, and R. Karri, “Veriinteresting: An empirical study of model prompt interactions in verilog code generation,” 2026. [Online]. Available: <https://arxiv.org/abs/2603.08715>
- [13] M. Alberti-Binimelis, C. Gutierrez-Gomez, D. Garcia-Gasulla, E. Parisi, R. Moundir Ghorab, O. Montenegro, B. Homs, M. Moreto, and G. Kestor, “Revisiting turtle: A comprehensive evaluation of llms for rtl generation,” *ACM Trans. Des. Autom. Electron. Syst.*, Jul. 2026, just Accepted. [Online]. Available: <https://doi.org/10.1145/3831369>
- [14] Y. Zhu, D. Huang, H. Lyu, X. Zhang, C. Li, W. Shi, Y. Wu, J. Mu, J. Wang, Y. Zhao, P. Jin, S. Cheng, S. Liang, X. Zhang, R. Zhang, Z. Du, Q. Guo, X. Hu, and Y. Chen, “Qimeng-codex-rl: Reasoning-enhanced verilog generation,” 2026. [Online]. Available: <https://arxiv.org/abs/2505.24183>
- [15] C. Deng, Y.-D. Tsai, G.-T. Liu, Z. Yu, and H. Ren, “Scalertl: Scaling llms with reasoning data and test-time compute for accurate rtl code generation,” 2025. [Online]. Available: <https://arxiv.org/abs/2506.05566>
- [16] L. Huang, R. Zhang, J. Guo, Y. Zhang, D. Huang, S. Cheng, P. Jin, C. Li, Z. Du, X. Hu, Y. Chen, and Q. Guo, “Qimeng-crux: Narrowing the gap between natural language and verilog via core refined understanding expression for circuit design,” 2026. [Online]. Available: <https://arxiv.org/abs/2511.20099>
- [17] F. Cui, C. Yin, K. Zhou, Y. Xiao, G. Sun, Q. Xu, Q. Guo, D. Song, D. Lin, X. Zhang, Yun, and Liang, “Origen:enhancing rtl code generation with code-to-code augmentation and self-reflection,” 2024. [Online]. Available: <https://arxiv.org/abs/2407.16237>
- [18] Y. Zhao, D. Huang, C. Li, P. Jin, M. Song, Y. Xu, Z. Nan, M. Gao, T. Ma, L. Qi, Y. Pan, Z. Zhang, R. Zhang, X. Zhang, Z. Du, Q. Guo, and X. Hu, “Codex: Empowering llms with hdl generation through multi-level summarization,” 2025. [Online]. Available: <https://arxiv.org/abs/2407.10424>
- [19] N. Wang, B. Yao, J. Zhou, Y. Hu, X. Wang, N. Guan, and Z. Jiang, “Insights from verification: Training a verilog generation llm with reinforcement learning with testbench feedback,” 2025. [Online]. Available: <https://arxiv.org/abs/2504.15804>
- [20] 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,” 2025. [Online]. Available: <https://arxiv.org/abs/2501.04908>
- [21] S. Liu, W. Fang, Y. Lu, Q. Zhang, H. Zhang, and Z. Xie, “Rtlcoder: Outperforming gpt-3.5 in design rtl generation with our open-source dataset and lightweight solution,” in *2024 IEEE LLM Aided Design Workshop (LAD)*. IEEE, 2024, p. 15. [Online]. Available: <http://dx.doi.org/10.1109/LAD62341.2024.10691788>
- [22] Y. Wang, G. Sun, W. Ye, G. Qu, and A. Li, “Verireason: Reinforcement learning with testbench feedback for reasoning-enhanced verilog generation,” 2025. [Online]. Available: <https://arxiv.org/abs/2505.11849>
- [23] G. Gerganov and contributors, “llama.cpp: Llm inference in c/c++,” <https://github.com/ggml-org/llama.cpp>, 2023.