Benchmarking Agentic HLS Design Tasks With HLS-Eval

Stefan Abi-Karam[1,2], Callie Hao[1]
[1]Georgia Institute of Technology, Atlanta, USA [2]Georgia Tech Research Institute, Atlanta, USA
stefanabikaram@gatech.edu, callie.hao@gatech.edu

***Abstract*—Large language models (LLMs) and AI agents are increasingly explored for hardware design, including high-level digital design. While most work targets code generation and editing for hardware description languages (HDLs), our prior work introduced HLS-Eval, an open-source benchmark for evaluating LLMs on high-level synthesis (HLS) design tasks. Those evaluations, however, focused on zero-shot generation and editing, leaving open how agents achieve HLS design tasks.**

We therefore extend HLS-Eval with an agentic evaluation flow built on the open-source mini-swe-agent framework. The flow lets HLS design agents use file-editing tools, invoke a C++ compiler for self-verification, and iteratively refine designs during inference, while logging agent traces for analysis of cost, token usage, and iteration count. We present initial results on the existing HLS-Eval benchmarks.

In our initial evaluation, we find open-source LLMs paired with an agentic harness solve every simple HLS code generation task in our evaluation, underscoring the need to expand benchmark difficulty as model capabilities advance. Analyzing traces from passing and failing runs, we show how model size, token usage, and trajectory length relate to design pass rates. These results establish a foundation for agentic HLS design and motivate harder benchmarks and new agentic tooling as model capabilities progress.

I. Introduction

Building domain-specific accelerators, such as for autonomous navigation and robotics [27], high-energy physics [11], [16], and AI inference [5], [6], [29] has traditionally required long design cycles and deep hardware expertise. FPGAs with high-level synthesis (HLS) help address this by enabling accelerators to be developed from high-level C, C++, or Python descriptions [6], [12], [14], [29]. By automating low-level tasks such as scheduling, binding, and dataflow generation, HLS promises to democratize domain-specific computing [8] by making hardware design more accessible to domain experts.

However, producing *high-performance* HLS accelerators still requires substantial expertise. Designers must apply directives such as loop unrolling, array partitioning, and pipelining [18], structure dataflow and streaming computations, explore large parameterized design spaces, and reason about how source-level choices affect post-synthesis latency and resource use. This creates a steep barrier for non-experts, especially across diverse vendor and academic HLS toolchains.

Large language models (LLMs) have shown strong potential in software engineering [7] and HDL-based hardware design [3], [15], [19], [22], [26], including code generation, optimization, and tool use. Yet comparable gains have not been realized for HLS [2], where current methods remain far from the performance needed for practical domain-expert adoption. Although agentic LLM systems can plan, invoke tools, and solve multi-step tasks, their application to HLS workflows is still early [10], [17], [21], [25], [30].

Building on this observation, we identify a primary limitation that we propose to address: the field lacks comprehensive agentic HLS design benchmarks. Existing benchmarks fail to capture the full complexity of realistic HLS design tasks or quantify agent efficiency (e.g., cost and runtime) relative to achieved design performance. This gap in benchmarking infrastructure limits systematic evaluation across the hardware design research community, and we attribute it as a key reason progress in agentic HLS automation has lagged behind LLM-driven RTL design.

- We extend our prior AI HLS design benchmarking work, HLS-Eval [2], with an agentic evaluation flow built on a simple agent harness, enabling standardized assessment of LLM agents on HLS design tasks.
- We evaluate this flow on HLS kernel generation from natural language descriptions and high-level specifications, reporting pass rates, inference scaling, and insights from agent traces.

These results offer a starting point for integrating and benchmarking the proposed ideas into a larger, more capable agentic HLS design flow as highlighted in Figure 1.

II. Methodology

We augment HLS-Eval with a new evaluation flow built on the mini-swe-agent harness [1], a simplified version of SWE-agent [28] widely used for reproducible LLM benchmarks on software engineering tasks. In this setup, the agent's only tool is a Bash shell inside a Linux Docker container, providing a minimal baseline with standard file utilities, C/C++ compilers, and scripting tools.

During evaluation, the agent receives an initial prompt and access to mounted design files within the container. After task completion or upon reaching the step or cost limits, we inspect the output directory for the final generated HLS implementation and verify that no provided testbench or header files were modified, preventing the agent from cheating on the evaluation.

We emphasize that this agentic flow directly subclasses the HLS-Eval `Evaluation` base class, and therefore inherits the rest of the HLS-Eval benchmarking infrastructure, including parallelization, automated HLS tool calls, and model and task parameterization. As a result, integrating the new flow, adapting our existing run scripts, and collecting our new results took less than one PhD-student hour.

Complete details of the implementation can be found in the HLS-Eval open-source code repository: `https://github.com/sharc-lab/hls-eval`.

III. Results

We evaluate our proposed agentic evaluation flow on the set of HLS designs already present within HLS-Eval: a total of 85 designs from the CHStone [13], MachSuite [24], Polybench [23], Rosetta [31], and C2HLSC [9] design sources. We chose to evaluate the `gpt-oss-20b` and `gpt-oss-120b` LLM models within our agentic harness. We selected these models

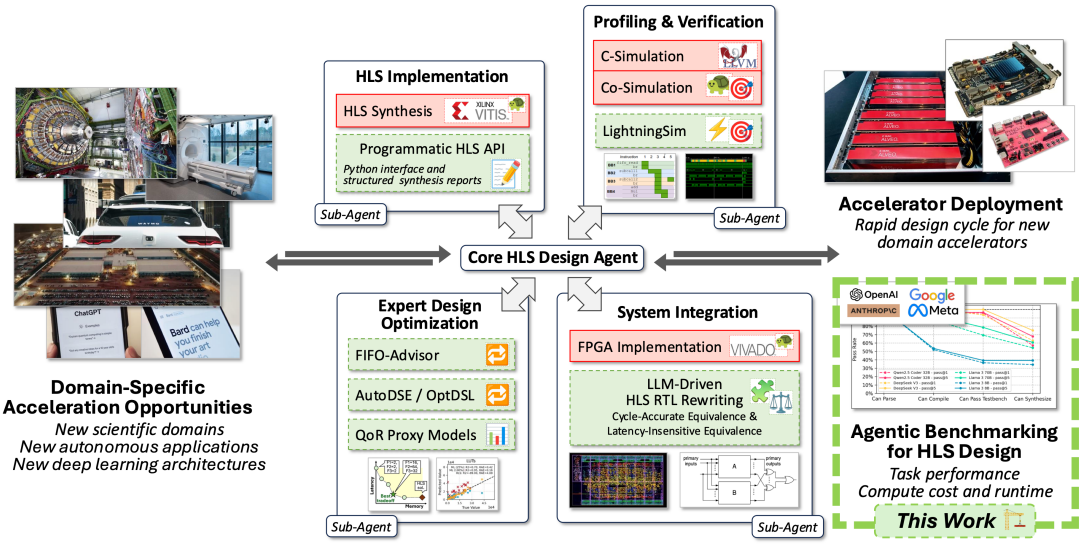

Fig. 1: Overview of our proposed end-to-end agentic HLS workflow for rapid prototyping of high-performance domain accelerators, with emphasis on extending HLS-Eval for benchmarking agentic systems for HLS design.

for our initial evaluation because they represent strong, low-cost, open-source baselines with two model size variations for comparison.

A. HLS Design Generation Task

We present pass rate results for the HLS design generation task in Figure 2. For each model, we evaluate the pass@k rate across multiple stages of design validation: parsing the design from the agent's output, checking whether the generated design compiles, checking whether the compiled design passes a functional testbench, and checking whether the HLS tool successfully synthesizes the design. We report pass@k for k=1 and k=10, with N=10 samples per evaluation case.

Most strikingly, `gpt-oss-120b` completely saturates the benchmark at the pass@10 rate. Its corresponding pass@1 rate also exceeds 90% for all HLS design stages. This shows that even a modestly sized open-source model, relative to larger commercial and open-source models, can generate simple kernels that are both synthesizable and correct, given only a natural-language specification and a testbench harness. We attribute this capability to the model's ability to call a C++ compiler for syntactic and functional self-verification at inference time, which highlights the key advantage of agents over naive zero-shot inference.

As a result, since saturated results provide no further informative feedback to measure AI models' HLS design capabilities, they motivate the development of more complex benchmark design cases that represent tasks akin to developers building full end-to-end domain accelerators.

B. Verifiers and Inference Scaling

Many stages of the HLS design flow act as strong verifiers for LLM-based agents, enabling self-checking of design correctness, functionality, and performance during inference. C++ compilers, co-simulation tools, HLS synthesis tools, design space exploration (DSE) frameworks, and downstream implementation tools serve as verifiers when accessible to the agent, and also enable inference scaling. Since only one agent

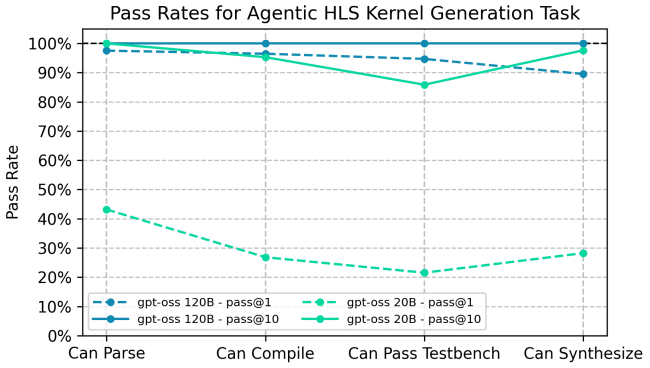

Fig. 2: Pass rates for mini-swe-agent at each stage of the HLS design flow. The `gpt-oss-120b` model saturates the benchmark with a 100% pass rate at all stages with $k = 10$.

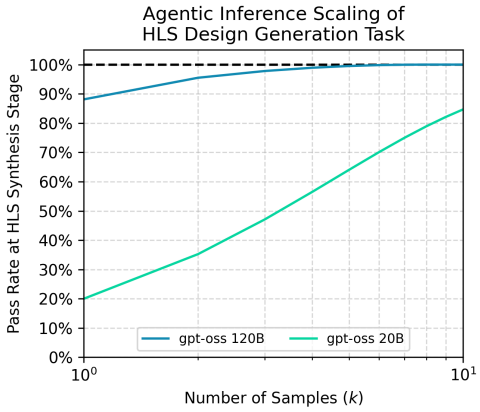

Fig. 3: Inference scaling results on pass rate for agentic HLS kernel generation task. The smaller model can recover +60% pass rate with $k = 10$ samples.

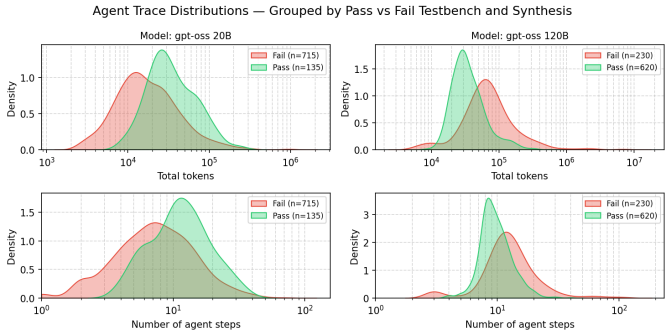

Fig. 4: Distribution of token counts and trajectory lengths for agent runs, grouped by model and evaluation outcome.

rollout needs to produce a design that passes all verification stages, sampling more agent rollouts for the same task can substantially improve pass rates [4] and overall design quality.

As shown in Figure 3, our agentic evaluations exhibit this inference scaling behavior. We observe large improvements from $k = 1$ samples to $k = 10$ samples, with gains depending on model size. The effect is strongest for weaker models: the `gpt-oss-20b` model achieves a $+60\%$ increase in synthesis pass rate at $k = 10$, while the larger `gpt-oss-120b` model sees a $+11\%$ improvement.

The sampling strategy described represents only one form of inference scaling. Future work will explore structured LLM-assisted evolutionary search [20] and methods that balance inference scaling benefits with model inference cost and tool runtime overhead.

C. Agent Trajectories

To better understand agent behavior on HLS design tasks, we present an initial analysis of inference traces. Figure 4 shows the distribution of token counts and trajectory steps, grouped by whether the design generated during a trace passed or failed the testbench and synthesis. We observe model-dependent trends: for the smaller `gpt-oss-20b` model, passing traces use more tokens and more steps, whereas for the larger `gpt-oss-120b` model, failing traces use more tokens and more steps.

Although the causes of these dynamics and their relationship to model scale and task complexity remain unclear, these findings motivate deeper investigation. Future work will incorporate more HLS-specific trace analysis, including tool usage frequency and correlations between tool runtimes and final design performance.

IV. Conclusion

We present our initial effort to extend HLS-Eval toward end-to-end agentic design of domain-specific accelerators using HLS. We report preliminary results on open-ended agentic evaluation for HLS design tasks, and we explore inference scaling and trace analysis in the context of agentic HLS design. By combining LLM-driven HLS agents, advanced tool automation, and standardized benchmarking within reproducible open-source frameworks, we aim to enable scalable, intelligent, and accessible domain acceleration for scientists, engineers, and researchers alike.

References

[1] "SWE-agent/mini-swe-agent," SWE-agent. [Online]. Available: https://github.com/SWE-agent/mini-swe-agent

[2] S. Abi-Karam and C. Hao, "HLS-eval: A benchmark and framework for evaluating LLMs on high-level synthesis design tasks," in *2025 IEEE International Conference on LLM-Aided Design (ICLAD)*. IEEE, pp. 219–226. [Online]. Available: https://ieeexplore.ieee.org/document/11106033/

[3] J. Blocklove, S. Thakur *et al.* Can EDA Tool Feedback Improve Verilog Generation by LLMs?

[4] B. Brown, J. Juravsky *et al.* Large Language Monkeys: Scaling Inference Compute with Repeated Sampling. [Online]. Available: http://arxiv.org/abs/2407.21787

[5] H. Chen, J. Zhang *et al.*, "Understanding the Potential of FPGA-based Spatial Acceleration for Large Language Model Inference," vol. 18, no. 1, pp. 5:1–5:29. [Online]. Available: https://dl.acm.org/doi/10.1145/3656177

[6] H. Chen, N. Zhang *et al.*, "Allo: A Programming Model for Composable Accelerator Design," vol. 8, pp. 171:593–171:620. [Online]. Available: https://dl.acm.org/doi/10.1145/3656401

[7] M. Chen, J. Tworek *et al.* Evaluating Large Language Models Trained on Code.

[8] Y. Chi, W. Qiao *et al.*, "Democratizing Domain-Specific Computing," vol. 66, no. 1, pp. 74–85. [Online]. Available: https://dl.acm.org/doi/10.1145/3524108

[9] L. Collini, S. Garg, and R. Karri, "C2HLSC: Leveraging Large Language Models to Bridge the Software-to-Hardware Design Gap." [Online]. Available: https://dl.acm.org/doi/10.1145/3734524

[10] L. Collini, A. Hennessee *et al.* Can Reasoning Models Reason about Hardware? An Agentic HLS Perspective. [Online]. Available: http://arxiv.org/abs/2503.12721

[11] J. Duarte, S. Han *et al.*, "Fast inference of deep neural networks in FPGAs for particle physics," vol. 13, no. 07, pp. P07 027–P07 027. [Online]. Available: http://arxiv.org/abs/1804.06913

[12] S. Fang, H. Chen *et al.* Dato: A Task-Based Programming Model for Dataflow Accelerators. [Online]. Available: http://arxiv.org/abs/2509.06794

[13] Y. Hara, H. Tomiyama *et al.*, "CHStone: A benchmark program suite for practical C-based high-level synthesis," in *2008 IEEE International Symposium on Circuits and Systems (ISCAS)*, pp. 1192–1195.

[14] S. Huang, K. Wu *et al.*, "PyLog: An Algorithm-Centric Python-Based FPGA Programming and Synthesis Flow," vol. 70, no. 12, pp. 2015–2028. [Online]. Available: https://ieeexplore.ieee.org/document/9591456

[15] F. R. Kashanaki, M. Zakharov, and J. Renau, "HDLEval Benchmarking LLMs for multiple HDLs," in *2024 IEEE LLM Aided Design Workshop (LAD)*, pp. 1–5.

[16] J. Kvapil, G. Borca-Tasciuc *et al.* Intelligent experiments through real-time AI: Fast Data Processing and Autonomous Detector Control for sPHENIX and future EIC detectors. [Online]. Available: http://arxiv.org/abs/2501.04845

[17] R. Li, J. Xiong *et al.* ChatHLS: Towards Systematic Design Automation and Optimization for High-Level Synthesis. [Online]. Available: http://arxiv.org/abs/2507.00642

[18] J. d. F. Licht, M. Besta *et al.* Transformations of High-Level Synthesis Codes for High-Performance Computing. [Online]. Available: http://arxiv.org/abs/1805.08288

[19] M. Liu, N. Pinckney *et al.*, "VerilogEval: Evaluating large language models for verilog code generation," in *2023 IEEE/ACM International Conference on Computer-Aided Design (ICCAD)*.

[20] A. Novikov, N. Vũ *et al.* AlphaEvolve: A coding agent for scientific and algorithmic discovery. [Online]. Available: http://arxiv.org/abs/2506.13131

[21] A. E. Oztas and M. Jelodari. Agentic-HLS: An agentic reasoning based high-level synthesis system using large language models (AI for EDA workshop 2024). [Online]. Available: http://arxiv.org/abs/2412.01604

[22] N. Pinckney, C. Batten *et al.*, "Revisiting VerilogEval: Newer llms, in-context learning, and specification-to-RTL tasks."

[23] L.-N. Pouchet and U. Bondugula, "PolyBench." [Online]. Available: https://web.cse.ohio-state.edu/~pouchet.2/software/polybench/

[24] B. Reagen, R. Adolf *et al.*, "MachSuite: Benchmarks for Accelerator Design and Customized Architectures," in *Proceedings of the IEEE International Symposium on Workload Characterization*, Raleigh, North Carolina, Oct. 2014.

[25] S. A. Sheikholeslam and A. Ivanov. SynthAI: A Multi Agent Generative AI Framework for Automated Modular HLS Design Generation. [Online]. Available: http://arxiv.org/abs/2405.16072

[26] S. Thakur, B. Ahmad *et al.*, "VeriGen: A Large Language Model for Verilog Code Generation," vol. 29, no. 3, pp. 1–31. [Online]. Available: https://dl.acm.org/doi/10.1145/3643681

[27] Z. Wan, A. S. Lele *et al.*, "Robotic computing on fpgas: Current progress, research challenges, and opportunities," pp. 291–295. [Online]. Available: https://api.semanticscholar.org/CorpusID:248811737

[28] J. Yang, C. E. Jimenez *et al.*, "SWE-agent: Agent-computer interfaces enable automated software engineering," in *The Thirty-eighth Annual Conference on Neural Information Processing Systems*, 2024. [Online]. Available: https://arxiv.org/abs/2405.15793

[29] H. Ye, C. Hao *et al.*, "ScaleHLS: A New Scalable High-Level Synthesis Framework on Multi-Level Intermediate Representation," in *2022 IEEE International Symposium on High-Performance Computer Architecture (HPCA)*, pp. 741–755. [Online]. Available: https://ieeexplore.ieee.org/abstract/document/9773203

[30] Z. Yu, M. Liu *et al.*, "Spec2RTL-Agent: Automated Hardware Code Generation from Complex Specifications Using LLM Agent Systems," in *2025 IEEE International Conference on LLM-Aided Design (ICLAD)*. IEEE, pp. 37–43. [Online]. Available: https://ieeexplore.ieee.org/document/11105957/

[31] Y. Zhou, U. Gupta *et al.*, "Rosetta: A Realistic High-Level Synthesis Benchmark Suite for Software Programmable FPGAs," in *Proceedings of the 2018 ACM/SIGDA International Symposium on Field-Programmable Gate Arrays*, ser. FPGA '18. Association for Computing Machinery, pp. 269–278. [Online]. Available: https://dl.acm.org/doi/10.1145/3174243.3174255