LLM4HLS-Agent: Budgeted Autonomous Repair and Multi-Objective Optimization for HLS

Duru Yağmur Akyol, Ce Guo, and Wayne Luk
Department of Computing, Imperial College London, London, United Kingdom
{yagmur.akyol25, c.guo, w.luk}@imperial.ac.uk

***Abstract*—Large language models (LLMs) can generate and modify high-level synthesis (HLS) code, but their proposals may fail functional checks or violate timing and resource constraints. We present *LLM4HLS-Agent*, an autonomous controller that accounts for model calls, tokens and synthesis-tool runs, within a shared budget. Before each proposal, the controller checks the remaining model allowance and validation runs. Verified candidates are assessed against timing and resource limits, while violations from promising but infeasible candidates guide further source changes. The controller retains verified designs, so budget exhaustion stops exploration without discarding an established result. On Alveo U55C with Vitis HLS 2025.2, we evaluated 20 generation, repair and optimization tasks with three models. The controller returned verified designs in 54 of 60 runs; optimization runs could return an unchanged baseline. For biconjugate gradient (BICG), feedback from a faster but resource-infeasible candidate guided recovery to a verified design with 61.8% lower HLS-estimated latency than the baseline while satisfying all configured timing and resource limits, within five model calls. These results demonstrate verified repair and optimization under explicit execution limits, without requiring an optimal budget allocation.**

***Index Terms*—high-level synthesis, large language models, budgeted search, autonomous repair, FPGA**

I. Introduction

HLS tools already combine repair with optimization [1], steer exploration by measured bottlenecks [2], and are judged by synthesis and resource use [3]. Running such a loop unattended needs its cost stated explicitly: one cost limit shared by repair and optimization, a rule for what may still be attempted as the search comes near the limit, and a defined result when the limit is reached.

Our contribution is a controller that accounts for search cost explicitly while preserving verified designs. The code is open source.[1]

II. Budget-Constrained Controller

Fig. 1 shows the controller. A task sets the source, a protected testbench, the top function, the target (part, clock, frequency floor and resource limits), the model, and the configured limits: a maximum number of model calls, an optional total-token cap, and caps on the number of C simulation (CSim), synthesis and C/RTL co-simulation runs. All model calls and validation runs count toward these limits, and a ledger records each. Before every model call the controller checks what remains: at least one model call and, for the tool runs the next candidate will need, one CSim run, one synthesis run and, if the task requires it, one co-simulation run. The token cap is checked only for remaining capacity, not for the size of the next response, so the final response may exceed it; the run then stops, that proposal is discarded before validation, and the last verified design is preserved and returned.

Our approach has two parts. In part A, CSim, synthesis and co-simulation run in order. When a stage fails, the model is asked for a fix guided by that stage's output, and validation restarts from CSim, because one repair can reveal a hidden fault. The source that passes every stage becomes the *verified baseline*, and its Vitis reports are read for the main bottleneck and its source cause. If no source passes every stage before the limits are reached, the run ends without a verified baseline and is reported as failed. Part B follows a configured schedule of exploration, refinement and recovery attempts. Each attempt consists of a proposal and its validation and may require more than one model call if the proposal fails the static checks. The schedule determines the kind of attempt, while measured results determine the parent candidate and strategy used within it. Every proposal passes static checks (unsafe edits, duplicates, no change) before CSim, synthesis and, when required, co-simulation, because the model's own account of its edit is not evidence. A verified candidate within the resource limits enters a Pareto set; one exceeding them does not enter the Pareto set, but its measured violations are returned to the model in the recovery attempt with the instruction to keep the gain and cut those resources.

The search ends when the configured attempt schedule is exhausted or when the budget check above fails. Selection is lexicographic among fully verified designs: frequency requirement met, then resource limits met, then lower HLS-estimated latency, then lower resource cost. The baseline competes under the same rule, so with no feasible candidate the best verified design by that order is returned, as the exact verified source rather than the model's latest output.

III. Evaluation Across Tasks and Models

All runs use the FPT'26 Track A toolchain and target (Vitis HLS 2025.2, Alveo U55C, 100 MHz). For the 60-run study, optimization used at most five attempts: three exploration attempts, one refinement attempt and one recovery attempt. A run is one execution of the controller on one task with one model, under that task's budget. To evaluate the capabilities separately, we fixed the execution mode for each task. The 14

Accepted to design competition track.

[1]https://github.com/duruyagmurakyol/llm4hls-agent

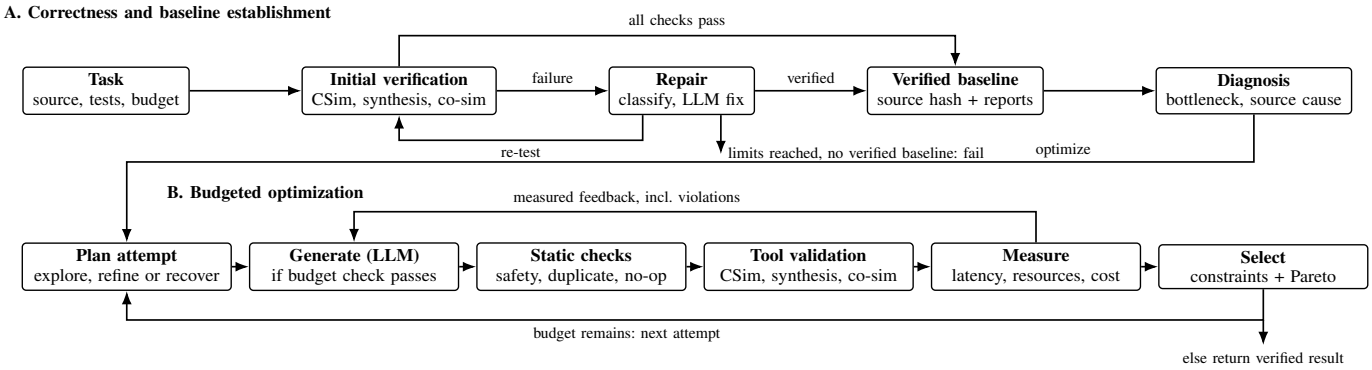

Fig. 1. LLM4HLS-Agent. Part A establishes a verified baseline and a diagnosis; part B follows the configured optimization-attempt schedule and returns the verified result when the limits are reached. The budget check at Generate requires a model call, the tool runs the next candidate needs, and token capacity.

TABLE I
TASKS, RUNS (ONE PER TASK PER MODEL) AND COMPLETED RUNS.

Task category	Tasks	Runs	Completed
Specification-to-kernel generation	1	3	3
Synthesis repair	1	3	3
Functional and interface repair	10	30	27
Structural repair (stream deadlock)	2	6	6
Latency and resource optimization	6	18	15
All	**20**	**60**	**54**

TABLE II
BICG ON THE U55C; LIMITS 34,964 LUT, 86,908 FF, 224 DSP.

Design	Latency (ns)	LUT	FF	DSP
Verified baseline (promoted)	7420.6	7,626	20,191	44
Pareto candidate (retained)	-40.7%	$+47.5\%$	$+17.2\%$	$+100\%$
Aggressive candidate (rejected)	3269.8	75,596	130,326	442
vs. baseline	-55.9%	$+891\%$	$+545\%$	$+905\%$
Recovered design (selected)	**2831.8**	24,938	52,102	176
vs. baseline	$\mathbf{-61.8\%}$	$+227\%$	$+158\%$	$+300\%$

generation and repair tasks ended when a verified baseline was established. The six optimization tasks required a valid starting design; sources that failed initial validation were rejected without repair. Seventeen tasks are ours (ATAX and GEMM from HLS-Eval [4]); three are organizer-supplied Track A tasks, not redistributed. Each of the 20 tasks was run once with each of three models (DeepSeek-V4-Pro, Qwen3.5-122B-A10B, Qwen3.6-27B), giving 60 runs (Table I) over nine kernels. A run is complete when it ends in a verified design; for optimization tasks that may be the unchanged baseline, so the optimization row of Table I counts completions, not improvements.

Every model completed the same 18 tasks; the six failed runs come from two tasks failing with all three models: an interface repair task (a wrong top-function name) that no model repaired, and the ATAX optimization task, whose supplied source failed initial validation and was rejected rather than repaired.

IV. EXAMPLE: ONE RUN ON BICG

Table II follows one run with Qwen3.5-122B-A10B on BICG, the PolyBench biconjugate-gradient kernel; italic rows are relative to the verified baseline, the target clock is 10 ns with a 100 MHz minimum, and latencies and frequencies are HLS estimates.

The run produced a Pareto candidate, verified and retained, and an aggressive candidate, faster but over every resource limit and therefore rejected. The recovery attempt then returned the aggressive candidate's measured violations to the model and produced the selected design, verified within all resource limits at an estimated 131.72 MHz. Under caps of five model calls, five CSim runs and four synthesis runs, the run used all five model calls, four CSim runs and four synthesis runs, totaling 14,925 tokens. Of the five model calls, two generated the aggressive candidate, the third performed recovery, and the final two produced the Pareto candidate.

V. CONCLUSION

One controller with explicit shared limits and a retained verified design completed 54 of 60 runs across three models; BICG shows rejection, recovery and return within five model calls. To make LLM4HLS-Agent useful for FPGA design automation, we plan to extend the approach to additional models and benchmarks and explore further strategies, including formal verification for safety-critical applications.

ACKNOWLEDGMENT

This work is supported by the United Kingdom EPSRC (grant numbers UKRI256, EP/V028251/1, EP/N031768/1, EP/S030069/1, and EP/X036006/1), AMD, SiliconFlow, Altera, and Intel.

REFERENCES

[1] R. Li, J. Xiong, X. He, J. Zhao, J. Lv, H. Fang, L. Qi, and X. Wang, "ChatHLS: Towards Systematic Design Automation and Optimization for High-Level Synthesis," in *Proc. ACL*, 2026, pp. 20996–21015.

[2] A. Sohrabizadeh, C. H. Yu, M. Gao, and J. Cong, "AutoDSE: Enabling Software Programmers to Design Efficient FPGA Accelerators," *ACM Trans. Des. Autom. Electron. Syst.*, vol. 27, no. 4, pp. 1–27, 2022.

[3] C. Guo and T. Zhao, "ResBench: A Resource-Aware Benchmark for LLM-Generated FPGA Designs," in *Proc. HEART*, 2025, pp. 25–34.

[4] S. Abi-Karam and C. Hao, "HLS-Eval: A Benchmark and Framework for Evaluating LLMs on High-Level Synthesis Design Tasks," in *Proc. IEEE ICLAD*, 2025, pp. 219–226.