

# Focus Session: Stepping Stones Towards Domain Acceleration Using AI-Driven High-Level Synthesis Design

Stefan Abi-Karam\*<sup>†</sup>, Miaoyan Zhou\*, Cong "Callie" Hao\*  
 \*Georgia Institute of Technology, <sup>†</sup>Georgia Tech Research Institute  
 {stefanabikaram, mzhou346, callie.hao}@gatech.edu

**Abstract**—Field-programmable gate arrays (FPGAs) combined with high-level synthesis (HLS) enable domain-specific accelerators from high-level algorithmic descriptions, but HLS has not yet democratized hardware design in practice. High-performance HLS design for accelerators still requires expert knowledge in HLS optimization, design space exploration (DSE), performance analysis, and toolflows with limited debuggability and analysis even for expert human designers.

To reduce this expertise burden, the community has explored ML-guided DSE, quality-of-results (QoR) proxy models, and LLMs for HLS code generation, editing, and optimization. However, progress remains constrained by scarce, accessible, high-quality HLS datasets and the difficulty of building diverse design collections. Many existing benchmarks sample design variations from a small pool of kernels rather than expanding the base pool of diverse designs. At the same time, most LLM hardware-design benchmarking emphasizes HDLs (e.g., Verilog), leaving comprehensive benchmarking infrastructure for HLS tasks comparatively limited. Furthermore, LLM-driven agents have shown strong automation and performance for software development tasks while agentic HLS design workflows still commonly lack robust interfaces to HLS tools, structured access to HLS reports, and feedback loops spanning HLS and downstream implementation.

To address these gaps, we present progress toward end-to-end rapid design of domain-specific accelerators using AI-driven approaches for HLS by targeting three problems: (1) HLS design datasets, (2) LLM benchmarking for HLS design, and (3) end-to-end agentic HLS design.

For HLS design datasets, we present HLSFactory, an end-to-end framework for building diverse HLS datasets that expands individual HLS designs into diverse, cross-vendor design spaces, runs HLS/FPGA toolflows at scale, and aggregates standardized outputs into packaged datasets for benchmarking and ML, deep learning, and LLM research. HLSFactory is API-driven and extensible, enabling users to contribute designs or results and plug in custom toolflows at any stage.

For LLM benchmarking, we present HLS-Eval, a benchmark and evaluation framework for LLM-driven HLS design that targets HLS code generation from natural language and HLS-specific optimization edits to existing code. It includes an "LLM-ready" benchmark for zero-shot evaluation, primarily drawing designs from HLSFactory for benchmarking.

Finally, we also present our ongoing effort toward an end-to-end agentic HLS design workflow that unifies accelerator design stages in a multi-agent framework, with integrated HLS design tools from our prior works including LightningSim (fast cycle-accurate simulation), FIFO-Advisor (automated HLS FIFO sizing), AutoDSE/OptDSL/HLSFactory (structured DSE for HLS), deep-learning proxy models (QoR prediction), and libvhls (programmable HLS tool interaction and hierarchical report analysis). Through this effort, we also plan to extend HLS-Eval to enable large-scale, parallel evaluations on realistic open-source HLS accelerators from academic projects, collecting agentic metrics on correctness, performance, agent runtime, and compute cost.

## I. INTRODUCTION

Field-programmable gate arrays (FPGAs) combined with high-level synthesis (HLS) offer the promise of democratizing domain-specific computing [1], enabling rapid development of accelerators for applications such as autonomous navigation [2], high-energy physics [3], [4], and AI inference [5]–[7] directly from high-level algorithmic specifications written in C, C++, or even Python [5], [6], [8], [9]. HLS tools handle low-level hardware details such as scheduling, resource binding, and dataflow implementation, allowing domain experts to focus on algorithmic optimization rather than cycle-by-cycle hardware design. However, despite these promises, HLS has not yet democratized hardware design in practice.

Rapidly deploying *high-performance* HLS accelerators still demands significant expertise across multiple dimensions. Designers must understand HLS-specific optimization directives such as loop unrolling, array partitioning, and pipelining; structure dataflow and streaming computations to meet compiler requirements (e.g., perfectly nested loops, compile-time bounds, single-producer single-consumer patterns); navigate design space exploration (DSE) across parameterized implementations; and interpret how source-code-level decisions affect post-synthesis metrics like latency and resource utilization. Furthermore, the specific subset of supported C++, available pragmas, compiler quirks, and specialized libraries differ among HLS tool vendors (e.g., Vitis HLS, Intel HLS Compiler, Catapult HLS), fragmenting the domain knowledge required for effective HLS design [10]. This expertise burden, combined with toolflows offering limited debuggability and analysis, creates a steep barrier for domain experts outside the expert HLS design community.

To reduce this expertise burden, the HLS community has explored multiple complementary approaches. Machine learning (ML) techniques have been proposed as part of HLS design workflows for quality-of-result (QoR) prediction and design space exploration [11]–[20]. Moving beyond traditional ML models, recent works have explored deep learning architectures [21]–[23] to capture complex relationships between HLS source code, intermediate representations, and target design metrics. Furthermore, large language models (LLMs) have demonstrated promise for HLS code generation, editing, and optimization [24]–[27], offering hope to aid designers in both writing efficient HLS designs from scratch and porting existing CPU- or GPU-targeted algorithms to FPGA platforms.

However, progress in ML-guided, deep learning-based, and LLM-driven HLS design remains constrained by three fundamental gaps:

(1) **Scarce, accessible, high-quality HLS datasets.** A key enabler to the success of ML, deep learning, and LLM approaches is high-quality datasets. While several open-source HLS datasets exist [22], [28]–[32], they suffer from fundamental limitations. First, these datasets are typically small or homogeneous, containing only subsets of published HLS benchmarks [29], [33]–[35] and frequently consisting exclusively of designs targeting a single vendor's HLS tool. Second, because of separately developed datasets, designs and intermediate/final tool outputs are organized in non-standard *ad hoc* ways with inconsistent reported metrics. Third, it remains challenging for external users to extend existing datasets due to missing details (e.g., tool versions, target devices, flow settings). Many existing works sample design variations from a small pool of kernels [22] rather than expanding the base pool of diverse designs. The fundamental limitation is not the lack of another complete HLS dataset, but rather the lack of a flexible and extensible framework enabling continuous community contributions to a standardized, sustainable dataset.

(2) **Limited LLM benchmarking infrastructure for HLS tasks.** While LLMs have shown strong results for hardware description language (HDL) design tasks [36]–[40], comprehensive benchmarking infrastructure for HLS tasks remains comparatively limited. Existing works on LLMs for HLS [25]–[27] explore different subsets of design tasks and benchmark sources but remain relatively limited in diversity and

lack holistic evaluation frameworks. There is a lack of large, diverse, open-source "LLM-for-HLS" benchmarks providing "LLM-ready" HLS designs for both code generation and code editing evaluations. Furthermore, there is no extensible software framework for researchers to create new benchmarks, integrate HLS tool interfaces for LLMs, and run parallel evaluations of code generation and editing tasks. Beyond just code generation, the real promise of LLMs for HLS is in hardware-optimizing code editing for automated iterative accelerator optimization.

**(3) Lack of robust end-to-end agentic HLS design workflows.** LLM-driven agents and agentic systems have enabled models to reason about multi-step plans and autonomously invoke tools for complex software development tasks. However, extending modern agentic systems to HLS design workflows remains in its infancy [41]–[45]. Existing LLM-driven HLS design workflows [25], [26], [46]–[51] provide limited or static feedback between design stages, restricting cross-stage optimization opportunities. Current agentic HLS systems lack deep integration with HLS tools, performance models, and feedback loops for iterative expert optimization. Furthermore, HLS-generated RTL often lacks usability and debuggability for integration with larger system designs and downstream FPGA implementation flows. Finally, there is a lack of comprehensive agentic HLS design benchmarks that capture realistic design complexity and quantify agent efficiency metrics such as cost and runtime relative to achieved design performance.

To address these gaps, we present progress toward end-to-end rapid design of domain-specific accelerators using AI-driven approaches for HLS by targeting all three problems: HLS design datasets, LLM benchmarking for HLS design, and end-to-end agentic HLS design. Our contributions include:

- **HLSFactory: End-to-end HLS dataset generation framework.** We present HLSFactory [52], our framework for HLS dataset generation, collection, expansion, and integration which encourages community contributions and user extensibility. HLSFactory provides an end-to-end compilation flow with three main stages: design space expansion to elaborate single parameterized HLS designs into large design spaces at the source-code level; design synthesis to run HLS and FPGA tools with sampled designs; and data aggregation to extract tool outputs for many designs into a standardized format. The framework is API-driven, extensible, and enables users to contribute designs or results and plug in custom toolflows at any stage. The included dataset covers diverse HLS designs with both simple designs synthesized across AMD/Xilinx and Intel toolflows with comprehensive metrics collection from functional simulation, HLS synthesis, and FPGA implementation.
- **HLS-Eval: Benchmark and evaluation framework for LLM-driven HLS design.** We present HLS-Eval [24], a comprehensive benchmark and evaluation framework for LLMs targeting HLS code generation from natural language and HLS-specific optimization edits to existing code. HLS-Eval includes "LLM-ready" designs sourced HLSFactory which includes designs from community HLS benchmarks, academic textbooks, and open-source hardware accelerators, each with testbenches, detailed natural language descriptions, and reference implementations. The framework provides an extensible Python API with HLS tool interfaces for LLMs, local and remote LLM inference, modular evaluator abstractions for user-defined inference flows, and a fine-grained parallel evaluation engine for large-scale benchmarking.
- **End-to-end agentic HLS design workflow.** We present our ongoing effort toward an end-to-end agentic HLS design workflow that unifies accelerator design stages in a multi-agent framework. The system integrates HLS design tools from our prior works including LightningSim for fast cycle-accurate simulation, FIFO-Advisor for automated HLS FIFO sizing, AutoDSE/OptDSL/HLSFactory for structured

DSE, deep-learning proxy models for QoR prediction, and libvhls for programmatic HLS tool interaction and hierarchical report analysis. Through this effort, we also plan to extend HLS-Eval to enable large-scale, parallel evaluations on realistic open-source HLS accelerators from academic projects, collecting agentic metrics on correctness, performance, agent runtime, and compute cost.

We direct readers to the HLSFactory and HLS-Eval publications for extensive details beyond what is described in following sections. Additionally, we develop these frameworks and tools as open-source at <https://github.com/sharc-lab>.

## II. DATASETS FOR HLS RESEARCH AND MACHINE LEARNING WITH HLSFACTORY

To overcome the hurdle of building high-quality, accessible HLS datasets, we present HLSFactory, a framework for constructing HLS datasets. We do not propose a new benchmark of designs or a fixed set of implemented design points. Instead, we provide a framework for collecting HLS implementation data, along with a base set of built-in designs and automation to parametrize and sample design variations. This choice shifts HLSFactory’s contribution away from raw dataset construction and toward features that help users from academic and industry communities to contribute new designs and use push-button automation to build ML-ready HLS datasets.

As illustrated in Fig. 1 the HLSFactory framework can process HLS designs to build complete HLS datasets in three stages.

**Stage 1: Design Space Expansion** elaborates parameterized HLS designs into large design spaces by enumerating combinations of HLS optimization compiler directives (i.e. pragmas). This allows users to create much larger and more diverse design datasets for ML models to generalize over. We introduce OptDSL, a vendor-agnostic domain-specific language for specifying design spaces through parameterized directives, including loop unrolling factors, array partitioning schemes, and pipeline configurations, which the design space expansion stage can sample to generate different parameterized HLS designs from a single design’s source code.

**Stage 2: Design Synthesis** executes HLS and FPGA toolflows in parallel across sampled designs. The framework supports AMD/Xilinx (Vitis HLS, Vivado) and Intel (i++, Quartus) toolflows, with a modular architecture enabling integration of additional vendors. A fine-grained parallel backend maximizes throughput by pooling tool calls across dataset collections rather than parallelizing within individual datasets.

**Stage 3: Data Aggregation** extracts standardized results from synthesis and implementation runs, packaging HLS-reported metrics (estimated latency, resource usage), post-implementation data (timing, power), and build artifacts into ML-ready formats. Users can contribute at any stage, submitting abstract designs with OptDSL specifications, concrete designs for direct synthesis, or pre-generated results for integration.

The HLSFactory framework includes built-in designs from community benchmarks (PolyBench, MachSuite, CHStone, Rosetta) totaling over 60 base designs, which can be expanded to thousands of design points through OptDSL parameterization. HLSFactory is also designed as a Python library, which allows users to load their own designs into these three main flows, add new built-in designs to HLSFactory, and define custom flows to process designs (e.g., simulation flows, custom analysis tools, etc.).

### A. Selected Case Studies

**ML-Based Post-Implementation QoR Prediction:** We demonstrate HLSFactory’s utility in hardware design optimization by building ML models for post-implementation QoR prediction, replicating the methodology of Dai et al. [12].

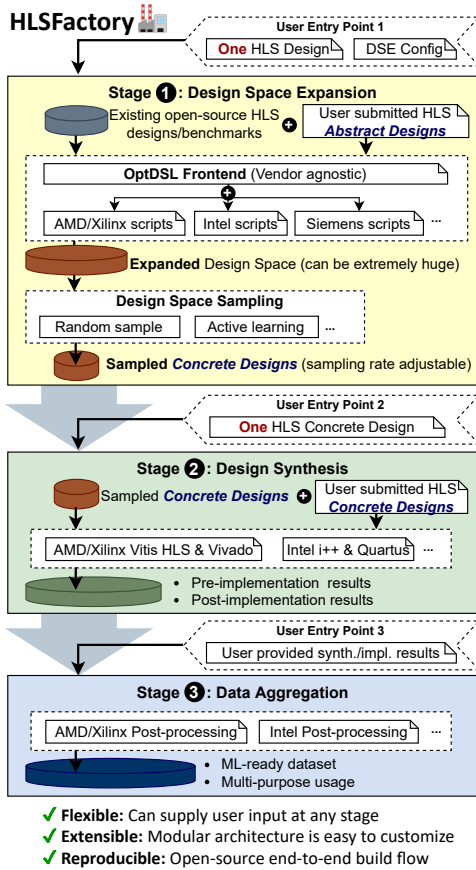


Fig. 1: A complete overview of the HLSFactory framework with three stages and three entry points where users can contribute their own designs.

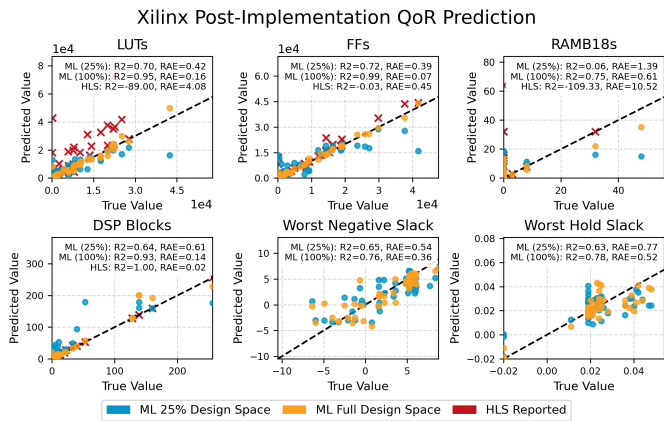


Fig. 2: True-vs-predicted plots for the HLS-based ML QoR model. Test values are shown for models trained on the complete and partial subset of the training design space. “RAE”: Relative Absolute Error ( $|\hat{y} - y|/|y - \bar{y}|$ ), “R2”: Coefficient of Determination

Using 29 base designs from PolyBench, MachSuite, and CHStone, we apply OptDSL-based design space expansion to generate  $n = 257$  concrete designs. HLSFactory’s APIs execute Vitis HLS synthesis and Vivado implementation, then aggregate results into tabular datasets.

We train histogram-based gradient boosting regression models to predict post-implementation resources (LUTs, FFs, BRAMs, DSPs) and timing metrics (worst negative slack,

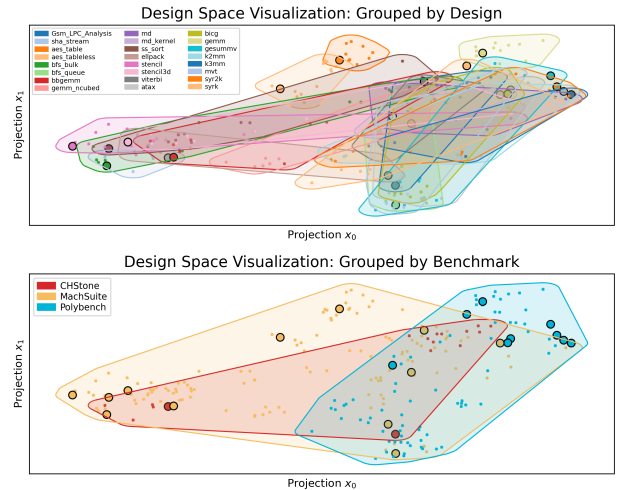


Fig. 3: Embedding of sampled designs across selected benchmarks. Base designs without optimizations are emphasized. Design points and locations are the same between both panels; they are only colored and grouped differently.

worst hold slack) using HLS-reported features as inputs. Fig 2 summarizes results on an 80%/20% train-test split, comparing models trained on the full expanded design space versus a 25% subset.

Training on the full design space consistently achieves higher  $R^2$  values and lower relative absolute error (RAE) across all prediction targets. Notably, for most targets, our ML models achieve lower RAE than the HLS tool’s own estimates. These results demonstrate both the utility of design space expansion for robust model training and the potential for ML to improve upon vendor tool predictions.

**Design Space Coverage Analysis:** We also qualitatively evaluate how design space expansion improves dataset diversity, which is critical for training ML models that generalize to unseen designs. Using PaCMAP dimensionality reduction to visualize the joint design space of HLS and post-implementation metrics, we observe in Fig. 3 that samples from different base designs occupy distinct, non-overlapping regions. The convex hulls around same-design samples show that expansion from even a small base set produces diverse, complementary coverage. This demonstrates that HLSFactory’s design space expansion approach effectively amplifies small sets of base designs into datasets suitable for training generalizable ML models.

### III. EVALUATING LLMs FOR HLS DESIGN TASKS WITH HLS-EVAL

To address the gap in benchmarking for HLS design tasks, mainly HLS code generation and optimization-driven code editing, we present HLS-Eval. Adopting a similar philosophy to HLSFactory, HLS-Eval is designed as a benchmarking framework that users can easily build and run LLM HLS benchmarks with user-desired designs, tasks, and inference techniques.

As shown in Fig. 4, we organize HLS-Eval into dataset construction, evaluation infrastructure, and the evaluations themselves.

**“LLM-ready” benchmark construction:** HLS-Eval includes  $\approx 90$  benchmark designs sourced from HLSFactory, spanning community HLS suites (PolyBench, MachSuite, CHStone, Rosetta) and additional academic/open-source accelerators components. Each design is normalized into an “LLM-ready” test case: (i) a header with types/macros and the top-level kernel signature, (ii) a kernel implementation file, (iii) a natural-language kernel description, and (iv) a self-contained C++ testbench for functional validation. To

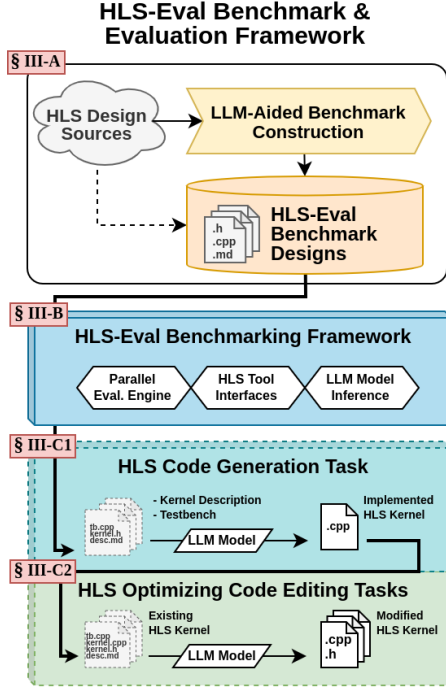


Fig. 4: Overview of HLS-Eval, which includes benchmark construction, an evaluation framework, and HLS design tasks such as code generation and optimization-based code editing evaluations.

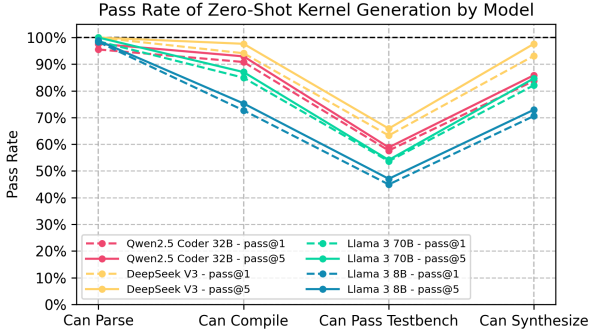


Fig. 5: Zero-Shot pass@k metrics for the HLS code generation task. We present four different pass@k metrics, one for each stage of HLS design.

scale benchmark construction across different design sources, HLS-Eval provides an optional semi-automated, LLM-aided CLI for hierarchy extraction, metadata-backed structured description generation, and testbench generation.

**Evaluation framework and parallel execution:** HLS-Eval provides a Python evaluation framework that unifies (i) LLM interfaces to the HLS toolchain, (ii) model backends for hosted inference, and (iii) a parallel evaluation engine with an extensible Evaluator API. We also provide python tool wrapper APIs for AMD/Xilinx Vitis HLS tools to provide LLM-friendly interfaces for C-simulation and HLS synthesis. Each tool call records return codes and captured outputs, enabling actionable feedback for iterative model-in-the-loop workflows. Additionally, HLS-Eval allows users to implement custom evaluators to test different inference techniques such as zero-shot, tool-feedback, and agentic evaluation.

#### A. Key Insights for HLS Designs with LLMs

**Zero-Shot Code Generation:** To establish baseline performance of open-source models on HLS code

generation, we evaluate the task of generating an HLS kernel implementation from a natural-language kernel description, a corresponding header file (types/macros and top-level signature), and a testbench. We assess outputs with four pass/fail metrics: parseability (extractable code blocks), compilability (C++ compilation under Vitis HLS), runnability (testbench passes with exit code 0), and synthesizability (successful HLS synthesis). The pass/fail metrics from each stage are aggregated as an unbiased pass@k [53] metric. All experiments use AMD/Xilinx Vitis HLS 2024.1 for C-simulation and synthesis, evaluate Llama 3 70B, Llama 8B, Qwen 2.5 Coder 32B, and DeepSeek V3 models, use  $N=5$  samples per design, and use  $k=1,5$  for pass rates.

We show our results in Fig. 5. The key takeaway from these code-generation results is that most models can generate valid, synthesizable HLS C++ code that is not functionally correct. We observe high pass rates for compilation and synthesis, but much poorer results for passing the testbench. As ongoing work, we treat these zero-shot results as baseline metrics for comparing other inference techniques, such as tool feedback, as well as more advanced open-source and commercial models.

**Tool Runtime Bottleneck:** We found that tool latency, mainly HLS synthesis, dominates benchmarking runtime. Average synthesis takes tens of seconds per design while LLM cloud inference, compilation, and simulation all are on the order of seconds. Naive “one design per thread” parallelism underutilizes resources because inference is fast but stalls behind HLS synthesis. HLS-Eval instead uses a fine-grained, stage-specific task-pool backend with independently tunable parallelism for inference, C-simulation, and HLS synthesis, maximizing utilization.

## IV. AGENTIC SYSTEMS FOR END-TO-END DESIGN AND DEPLOYMENT OF DOMAIN ACCELERATORS

Our ongoing work focuses on agentic HLS design. The key idea is to enable LLM-driven AI agents to possess the expert tools and knowledge of an experienced HLS designer, allowing them to rapidly prototype high-performance, domain-specific accelerators. We present a multi-agent approach that integrates several novel HLS tools.

As shown in Fig. ??, we envision a core LLM design agent orchestrates several specialized sub-agents for design implementation, expert optimization, profiling and Verification, and system integration, each equipped with tool-calling capabilities for domain-specific reasoning. These agents leverage fast cycle-accurate simulation, deep learning proxy models, and design space exploration engines, achieving design proficiency that can exceed both vendor tools and expert human designers.

#### A. Novel HLS Tool Integration with AI Agents

Naive LLM agents typically rely on file I/O and shell commands, which is not optimal for interacting with HLS code, HLS vendor tools, and hierarchical HLS design reports. We instead equip agents with domain-specific HLS tools, QoR proxy models, and programmatic interfaces. Equipping LLMs with these tools matches what expert HLS designers use to gain insight from simulation, design optimization, and report/trace analysis. We outline below a set of HLS-specific tools developed in prior work that we are integrating with LLM agents.

**Fast Cycle-Accurate Simulation:** We integrate LightningSim [54]–[56], a fast cycle-accurate HLS simulator with >99.9% accuracy that runs in seconds versus minutes to hours for C/RTL co-simulation. Unlike estimated latency in synthesis reports that can be inaccurate [57] or not possible to estimate, LightningSim provides exact latency measurements for testbenches with representative workloads. LightningSim’s CLI and Python API would enable agent-driven simulation, hierarchical trace analysis, and workload-guided optimization.

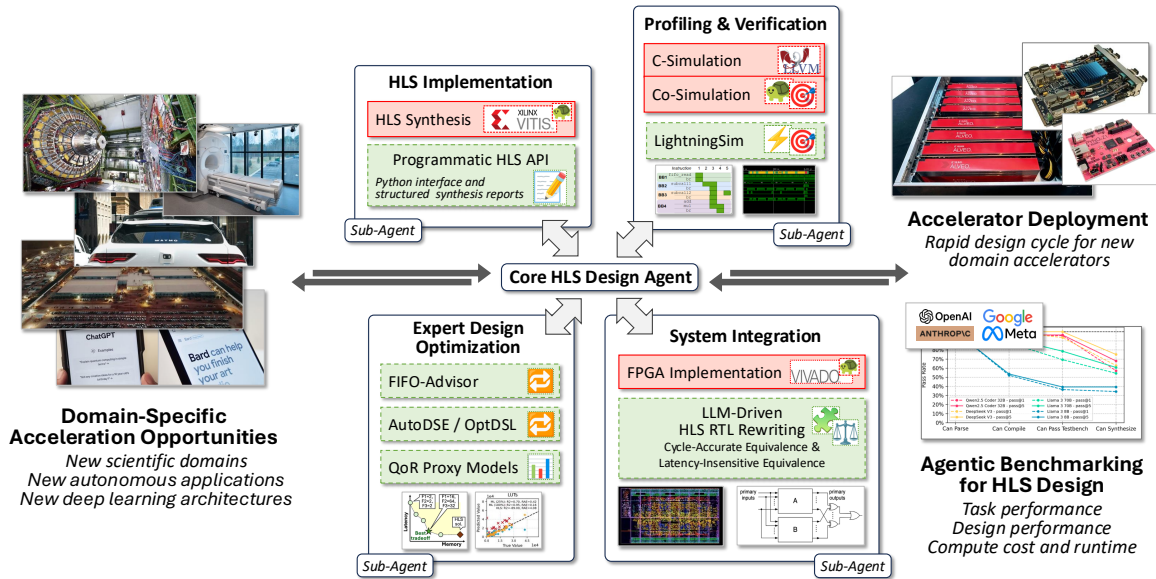


Fig. 6: Overview of the proposed end-to-end agentic HLS workflow for rapid prototyping of high-performance domain accelerators. The system features a core HLS design agent that coordinates specialized sub-agents, each responsible for distinct stages of the HLS flow and tool integration. This agentic architecture closes the optimization loop, where results from any sub-agent or tool feed back into the workflow to guide iterative refinement and accelerate design convergence.

**Automated FIFO Sizing:** Dataflow HLS designs use stream-based FIFOs. FIFO depth controls kernel latency and resources usage and must be sized properly avoid deadlocks. Shallow FIFOs stall on blocking accesses but use less resources, while deep FIFOs waste BRAM but result in faster kernels. We integrate FIFO-Advisor [58], which leverages LightningSim and black-box optimization to derive Pareto-optimal, deadlock-free FIFO configurations, exposed via both CLI and Python for agent execution and selection of Pareto points.

**QoR Proxy Models:** HLS synthesis is slow (minutes per build) and can yield approximate latency [57] and resource reports [52]. To counter this, prior works have explored pre-trained deep-learning QoR predictors [21], [22], [59], [60] (LLM/GNN-based) that estimate latency and FPGA resource utilization directly from source code and pragmas, enabling sub-second feedback for latency/resource-aware edits and generalizing across designs and tool versions. These models can also be retrained using our expanded HLSFactory dataset [52], spanning curated academic designs and synthetic HLS programs to improve diversity and generalization.

**Design Space Exploration Engine:** We equip agents with AutoDSE [61] and OptDSL within HLSFactory [52] to explore compiler pragmas and compile-time parameter spaces. AutoDSE performs bottleneck-driven DSE, prioritizing configurations with the highest performance impact. OptDSL specifies the design space of configurations for a given HLS design. Agents can generate OptDSL design spaces and delegate exploration to return Pareto-optimal design configurations, accelerated by QoR proxy models that reduce per-point evaluation from minutes to seconds, as shown in prior work [21].

**Programmatic HLS Tool Interface:** Because commercial HLS tools provide limited automation APIs, we extend our open-source libvhls to give agents programmatic control over builds, script generation, report parsing, and hierarchical analysis. This enables parallel synthesis at scale (many-core machines/remote clusters), multi-report inspection, hierarchy-level bottleneck tracing, and out-of-context HLS synthesis.

### B. Agentic Benchmarks for Complex HLS Design Tasks

Ongoing efforts also focus on extending our HLS-Eval benchmark to address the limited scope of existing LLM-aided

HLS benchmarks, which fail to capture the complexity of real-world, multi-tool design workflows. The expanded benchmark set will incorporate larger and more diverse open-source domain accelerator designs from academic groups. Unlike traditional community benchmarks, accelerators provide realistic scale and complexity, aligning with our goal of accelerating domain-specialized compute prototyping. To support agentic benchmarking, we also evaluate both conventional metrics (functional correctness and synthesizability pass rates) as well as agentic metrics (task length, inference cost, tool-calling and runtime statistics, and final design performance), enabling comprehensive evaluation of agentic efficiency relative to design complexity across open-source and commercial LLMs.

## V. CONCLUSION

We present three main stepping stones toward delivering end-to-end agentic design of domain-specific accelerators using HLS. We introduce HLSFactory and HLS-Eval to address two major hurdles: the lack of high-quality HLS dataset infrastructure and the lack of robust infrastructure for benchmarking LLM-driven HLS workflows. Both efforts emphasize community contribution and extensibility rather than relying solely on explicit data collection and rigid evaluation pipelines. Finally, we present our ongoing work on integrating novel optimization, analysis, and automation tools for HLS into a multi-agent system, with the goal of evaluating agentic performance on complex, domain-scale accelerator design benchmarks. By integrating LLM-driven HLS agents, advanced tool automation, and standardized benchmarking within reproducible open-source frameworks, we seek to enable scalable, intelligent, and accessible domain acceleration for scientists, engineers, researchers, and students alike.

## REFERENCES

- [1] Y. Chi, W. Qiao, A. Sohrabizadeh *et al.*, “Democratizing Domain-Specific Computing,” vol. 66, no. 1, pp. 74–85.
- [2] Z. Wan, A. S. Lele, B. Yu *et al.*, “Robotic computing on fpgas: Current progress, research challenges, and opportunities,” pp. 291–295.
- [3] J. Duarte, S. Han, P. Harris *et al.*, “Fast inference of deep neural networks in FPGAs for particle physics,” vol. 13, no. 07, pp. P07 027–P07 027.
- [4] J. Kvapil, G. Borca-Tasciuc, H. Bossi *et al.* Intelligent experiments through real-time AI: Fast Data Processing and Autonomous Detector Control for sPHENIX and future EIC detectors.

- [5] H. Ye, C. Hao, J. Cheng *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.
- [6] H. Chen, N. Zhang, S. Xiang *et al.*, "Allo: A Programming Model for Composable Accelerator Design," vol. 8, pp. 171:593–171:620.
- [7] H. Chen, J. Zhang, Y. Du *et al.*, "Understanding the Potential of FPGA-based Spatial Acceleration for Large Language Model Inference," vol. 18, no. 1, pp. 5:1–5:29.
- [8] S. Huang, K. Wu, H. Jeong *et al.*, "PyLog: An Algorithm-Centric Python-Based FPGA Programming and Synthesis Flow," vol. 70, no. 12, pp. 2015–2028.
- [9] S. Fang, H. Chen, N. Zhang *et al.* Dato: A Task-Based Programming Model for Dataflow Accelerators.
- [10] J. Fine Licht, M. Besta, S. Meierhans *et al.*, "Transformations of High-Level Synthesis Codes for High-Performance Computing," vol. 32, no. 5, pp. 1014–1029.
- [11] H. Mohammadi Makrani, F. Farahmand, H. Sayadi *et al.*, "Pyramid: Machine learning framework to estimate the optimal timing and resource usage of a high-level synthesis design," in *2019 29th International Conference on Field Programmable Logic and Applications (FPL)*, 2019.
- [12] S. Dai, Y. Zhou, H. Zhang *et al.*, "Fast and accurate estimation of quality of results in high-level synthesis with machine learning," in *2018 IEEE 26th Annual International Symposium on Field-Programmable Custom Computing Machines (FCCM)*. IEEE, 2018, pp. 129–132.
- [13] D. Liu and B. C. Schafer, "Efficient and reliable high-level synthesis design space explorer for fpgas," in *2016 26th International Conference on Field Programmable Logic and Applications (FPL)*, 2016.
- [14] W. Haaswijk, E. Collins, B. Seguin *et al.*, "Deep learning for logic optimization algorithms," in *2018 IEEE International Symposium on Circuits and Systems (ISCAS)*, 2018.
- [15] Y. Luo, C. Tan, N. B. Agostini *et al.*, "MI-cgra: An integrated compilation framework to enable efficient machine learning acceleration on cgras," in *2023 60th ACM/IEEE Design Automation Conference (DAC)*.
- [16] V. A. Chhabria, Y. Zhang, H. Ren *et al.*, "Mavirec: MI-aided vectored ir-drop estimation and classification," in *2021 Design, Automation & Test in Europe Conference & Exhibition (DATE)*, 2021.
- [17] R. G. Kim, J. R. Doppa, and P. P. Pande, "Machine learning for design space exploration and optimization of manycore systems," in *2018 IEEE/ACM International Conference on Computer-Aided Design (ICCAD)*, 2018, pp. 1–6.
- [18] Z. Lin, J. Zhao, S. Sinha *et al.*, "HL-Pow: A Learning-Based Power Modeling Framework for High-Level Synthesis," in *25th Asia and South Pacific Design Automation Conference (ASP-DAC)*, 2020.
- [19] G. Singha, D. Diamantopoulos, J. Gómez-Luna *et al.*, "LEAPER: Fast and Accurate FPGA-based System Performance Prediction via Transfer Learning," in *IEEE 40th International Conference on Computer Design (ICCD)*, 2022.
- [20] Z. Lin, Z. Yuan, J. Zhao *et al.*, "Powergear: Early-stage power estimation in fpga hls via heterogeneous edge-centric gnns," in *Design, Automation & Test in Europe Conference & Exhibition (DATE)*, 2022.
- [21] A. Sohrabzadeh, Y. Bai, Y. Sun *et al.*, "Robust GNN-Based Representation Learning for HLS," in *2023 IEEE/ACM International Conference on Computer Aided Design (ICCAD)*, pp. 1–9.
- [22] Y. Bai, A. Sohrabzadeh, Z. Qin *et al.*, "Towards a comprehensive benchmark for high-level synthesis targeted to FPGAs," in *Proceedings of the 37th International Conference on Neural Information Processing Systems*, ser. NIPS '23. Curran Associates Inc.
- [23] Y. Bai, A. Sohrabzadeh, Z. Ding *et al.*, "Learning to Compare Hardware Designs for High-Level Synthesis," in *Proceedings of the 2024 ACM/IEEE International Symposium on Machine Learning for CAD*, ser. MLCAD '24. New York, NY, USA: Association for Computing Machinery, Sep. 2024, pp. 1–7.
- [24] 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.
- [25] L. Collini, S. Garg, and R. Karri, "C2HLS: Leveraging Large Language Models to Bridge the Software-to-Hardware Design Gap."
- [26] C. Xiong, C. Liu, H. Li *et al.*, "HLSpilot: LLM-based High-Level Synthesis," in *Proceedings of the 43rd IEEE/ACM International Conference on Computer-Aided Design*, ser. ICCAD '24. Association for Computing Machinery, pp. 1–9.
- [27] J. Gai, Hao, Chen *et al.* Exploring Code Language Models for Automated HLS-based Hardware Generation: Benchmark, Infrastructure and Analysis.
- [28] Q. Gautier, A. Althoff, P. Meng *et al.*, "Spector: An opencl fpga benchmark suite," in *2016 International Conference on Field-Programmable Technology (FPT)*. IEEE, 2016, pp. 141–148.
- [29] Y. Zhou, U. Gupta, S. Dai *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.
- [30] Z. Wei, A. Arora, R. Li *et al.*, "HLSDataset: Open-source dataset for ML-assisted FPGA design using high level synthesis," in *2023 IEEE 34th International Conference on Application-specific Systems, Architectures and Processors (ASAP)*. IEEE, pp. 197–204.
- [31] L. Ferretti, J. Kwon, G. Ansaloni *et al.* DB4HLS: A Database of High-Level Synthesis Design Space Explorations.
- [32] P. Goswami, M. Shahshahani, and D. Bhatia, "MLSBench: A Synthesizable Dataset of HLS Designs to Support ML Based Design Flows," in *Proceedings of the 2020 ACM/SIGDA International Symposium on Field-Programmable Gate Arrays*, ser. FPGA '20. Association for Computing Machinery, p. 312.
- [33] L.-N. Pouchet and U. Bondugula, "PolyBench."
- [34] Y. Hara, H. Tomiyama, S. Honda *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.
- [35] B. Reagen, R. Adolf, Y. S. Shao *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.
- [36] F. R. Kshanaki, M. Zakharov, and J. Renau, "HDLLevel Benchmarking LLMs for multiple HDLs," in *2024 IEEE LLM Aided Design Workshop (LAD)*, pp. 1–5.
- [37] S. Thakur, B. Ahmad, H. Pearce *et al.*, "VeriGen: A Large Language Model for Verilog Code Generation," vol. 29, no. 3, pp. 1–31.
- [38] M. Liu, N. Pinckney, B. Khailany *et al.*, "VerilogEval: Evaluating large language models for verilog code generation," in *2023 IEEE/ACM International Conference on Computer-Aided Design (ICCAD)*.
- [39] N. Pinckney, C. Batten, M. Liu *et al.*, "Revisiting VerilogEval: Newer llms, in-context learning, and specification-to-RTL tasks."
- [40] J. Blocklove, S. Thakur, B. Tan *et al.* Can EDA Tool Feedback Improve Verilog Generation by LLMs?
- [41] 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).
- [42] R. Li, J. Xiong, X. He *et al.* ChatHLS: Towards Systematic Design Automation and Optimization for High-Level Synthesis.
- [43] L. Collini, A. Hennessee, R. Karri *et al.* Can Reasoning Models Reason about Hardware? An Agentic HLS Perspective.
- [44] Z. Yu, M. Liu, M. Zimmer *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.
- [45] S. A. Sheikholeslam and A. Ivanov. SynthAI: A Multi Agent Generative AI Framework for Automated Modular HLS Design Generation.
- [46] K. Xu, G. L. Zhang, X. Yin *et al.*, "HLSRewriter: Efficient Refactoring and Optimization of C/C++ Code with LLMs for High-Level Synthesis."
- [47] Q. Zou, N. Chen, Y. Chen *et al.* HLStrans: Dataset for LLM-Driven C-to-HLS Hardware Code Synthesis.
- [48] R. Li, J. Xiong, and X. Wang. iDSE: Navigating Design Space Exploration in High-Level Synthesis Using LLMs.
- [49] H. Wang, X. Wu, Z. Ding *et al.* LLM-DSE: Searching Accelerator Parameters with LLM Agents.
- [50] K. Xu, G. L. Zhang, X. Yin *et al.*, "Automated C/C++ Program Repair for High-Level Synthesis via Large Language Models," in *Proceedings of the 2024 ACM/IEEE International Symposium on Machine Learning for CAD*, ser. MLCAD '24. Association for Computing Machinery, pp. 1–9.
- [51] X. Yao, W. Zhao, Q. Sun *et al.*, "High-level Synthesis Directives Design Optimization via Large Language Model," vol. 30, no. 5, pp. 78:1–78:24.
- [52] S. Abi-Karam, R. Sarkar, A. Seigler *et al.*, "HLSFactory: A Framework Empowering High-Level Synthesis Datasets for Machine Learning and Beyond," in *2024 ACM/IEEE 6th Symposium on Machine Learning for CAD (MLCAD)*. Salt Lake City (Snowbird), UT, USA: IEEE, Sep. 2024, pp. 1–9.
- [53] M. Chen, J. Tworek, H. Jun *et al.* Evaluating Large Language Models Trained on Code.
- [54] R. Sarkar and C. Hao, "LightningSim: Fast and Accurate Trace-Based Simulation for High-Level Synthesis," in *2023 IEEE 31st Annual International Symposium on Field-Programmable Custom Computing Machines (FCCM)*. Marina Del Rey, CA, USA: IEEE, May 2023, pp. 1–11.
- [55] R. Sarkar, R. Paul, and C. C. Hao, "LightningSimV2: Faster and Scalable Simulation for High-Level Synthesis via Graph Compilation and Optimization," in *2024 IEEE 32nd Annual International Symposium on Field-Programmable Custom Computing Machines (FCCM)*, May 2024, pp. 104–114, iSSN: 2576-2621.
- [56] R. Sarkar and C. Hao. OmniSim: Simulating Hardware with C Speed and RTL Accuracy for High-Level Synthesis Designs.
- [57] J. Kim and C. Hao, "RealProbe: An Automated and Lightweight Performance Profiler for In-FPGA Execution of High-Level Synthesis Designs," in *2025 IEEE 33rd Annual International Symposium on Field-Programmable Custom Computing Machines (FCCM)*, pp. 10–18.
- [58] S. Abi-Karam, R. Sarkar, S. Basalama *et al.*, "FIFOAdvisor: A DSE Framework for Automated FIFO Sizing of High-Level Synthesis Designs," in *31st Asia and South Pacific Design Automation Conference (ASP-DAC)*.
- [59] N. Wu, H. Yang, Y. Xie *et al.*, "High-level synthesis performance prediction using GNNs: Benchmarking, modeling, and advancing," in *Proceedings of the 59th ACM/IEEE Design Automation Conference*, ser. DAC '22. Association for Computing Machinery, pp. 49–54.
- [60] E. Murphy and L. Josipović, "Balor: HLS Source Code Evaluator Based on Custom Graphs and Hierarchical GNNs," in *Proceedings of the 43rd IEEE/ACM International Conference on Computer-Aided Design*, ser. ICCAD '24. Association for Computing Machinery, pp. 1–9.
- [61] A. Sohrabzadeh, C. H. Yu, M. Gao *et al.*, "AutoDSE: Enabling Software Programmers to Design Efficient FPGA Accelerators," vol. 27, no. 4, pp. 32:1–32:27.