# LLM-Driven High-Level Synthesis Optimization for Application-Specific Integrated Circuits

Paride Farnese
Department of Computer Science and Engineering, University at Buffalo, Buffalo, NY, USA.
farneseparide@buffalo.edu

Isidro Dalmau
Department of Computer Science, Colorado State University, Fort Collins, CO, USA.
dalmauisidro@colostate.edu

Quinten Cornelissen
Department of Computer Science, University of Central Florida, Orlando, FL, USA.
qcornelissen@ucf.edu

**Abstract**

High-level synthesis has long promised to raise the abstraction level of application-specific integrated circuit design, yet practical adoption remains constrained by the difficulty of mapping behavioral intent to efficient power, performance, and area outcomes. Large language models introduce a new socio-technical layer into this process: they can generate, refactor, explain, and search hardware descriptions, but they also introduce uncertainty, provenance ambiguity, and infrastructure dependency. This paper examines LLM-driven high-level synthesis optimization for application-specific integrated circuits from a system-level perspective. It argues that the central challenge is not simply model accuracy but the co-design of model capability, compiler and synthesis feedback, verification infrastructure, data governance, and deployment economics. The paper reviews foundational high-level synthesis and electronic design automation work, then develops an architecture in which language models act as interface agents, transformation proposers, and design-space explorers rather than autonomous replacements for synthesis tools. It analyzes structural trade-offs among abstraction, predictability, resource efficiency, verification burden, and organizational accountability. It further considers robustness under distribution shift, fairness in access to models and compute, sustainability of repeated inference, and policy implications for safety-critical silicon. Case illustrations from ASIC, FPGA, and embedded systems contexts show that LLM-driven optimization is most credible when tightly coupled to formal checks, simulation, and human review. The conclusion outlines a governance-aware research agenda for reliable, measurable, and equitable LLM-assisted ASIC design.

**Keywords**

high-level synthesis, application-specific integrated circuits, large language models, electronic design automation, design-space exploration, verification, governance, sustainability.

## 1. Introduction

Application-specific integrated circuits occupy a distinctive position in computing because they convert algorithmic intent into fixed silicon structures whose efficiency, cost, and reliability are determined long before deployment. High-level synthesis is the engineering discipline that mediates this conversion by translating behavioral descriptions, often

expressed in languages such as C, C++, SystemC, or domain-specific dialects, into register-transfer-level implementations [1], [2], [3]. The promise of high-level synthesis is that designers can explore architecture, pipeline depth, memory organization, and interface structure without manually rewriting every control path and data path. The reality is more complicated. Synthesis results depend on coding style, directive selection, technology libraries, timing constraints, and tool-specific heuristics, so the path from behavioral code to optimized hardware remains partially opaque even to experienced engineers [4], [5].

Large language models have recently entered this space as assistants capable of generating hardware description language code, suggesting synthesis directives, explaining tool errors, and proposing alternative microarchitectures. Their appeal lies in their ability to operate across natural language, code, and documentation, which makes them plausible mediators between human design intent and machine-oriented synthesis flows [11], [12], [13]. At the same time, language models do not possess intrinsic knowledge of timing closure, power integrity, or physical layout. They predict plausible sequences of tokens, and plausibility in source code does not guarantee synthesizability, equivalence, or manufacturability. The research problem is therefore not whether a model can produce Verilog or VHDL, but whether an LLM-driven system can improve high-level synthesis outcomes under real constraints of verification, schedule, energy, and organizational trust.

This paper positions LLM-driven high-level synthesis optimization as a system-level problem. It examines how models, synthesis tools, verification engines, data pipelines, and human review processes must be arranged to produce reliable ASIC designs. It considers architectural trade-offs, infrastructure requirements, robustness under adversarial or distributional uncertainty, fairness in access to computational resources, sustainability of iterative inference, and policy implications for safety-critical silicon. The central claim is that LLM-driven optimization is most credible when treated as a bounded, auditable, and feedback-rich layer within electronic design automation rather than as a replacement for synthesis expertise. The paper proceeds through foundations, architecture, trade-offs, verification, governance, deployment, evaluation, and policy before concluding with a research agenda.

## 2. Foundations and Related Work

High-level synthesis has evolved from early scheduling and binding research into a mature but still constrained industrial practice [1], [2]. Foundational work established the core transformations: operation scheduling, resource allocation, module binding, controller generation, and interface synthesis. Later research expanded the scope to include loop transformations, memory partitioning, pipelining, and dataflow optimization, drawing on compiler transformation theory and parallel computing [7]. Contemporary high-level synthesis tools provide directive-driven control over latency, initiation interval, array mapping, and interface protocols, but they often require expert tuning to achieve competitive quality of results [3], [4]. This expert dependency creates a natural opening for machine learning and language-model assistance, because much of the tuning knowledge is encoded in informal design practice, tool reports, and iterative experimentation.

Field-programmable gate array research provided an important testbed for high-level synthesis because reconfigurable fabrics tolerate faster design iteration than ASIC flows [5], [6]. Tools such as LegUp demonstrated that processor-accelerator systems could be generated from high-level code, while surveys of FPGA high-level synthesis documented persistent gaps between productivity gains and quality-of-results expectations [4], [6]. These gaps

include difficulty in expressing parallel memory behavior, unpredictable timing after place and route, and limited support for legacy or mixed-abstraction designs. For ASICs, the stakes are higher because non-recurring engineering costs, mask costs, and irreversibility increase the penalty for late discovery of functional or timing defects. Consequently, LLM-driven ASIC optimization must satisfy stricter verification and traceability requirements than many software-oriented code generation tasks.

Hardware and software co-design research further clarifies why language models cannot be evaluated in isolation. Embedded and cyber-physical systems require joint consideration of computation, communication, physical constraints, and real-time behavior [8], [9]. Design decisions that improve one dimension, such as throughput, may degrade another, such as energy or predictability. Language models can help generate alternatives, but the evaluation function must remain grounded in synthesis reports, simulation, formal verification, and physical design metrics. Recent work on LLM-based Verilog generation has begun to address this grounding problem. Benchmarks and domain-adapted models have improved functional correctness on small designs [11], [12], [13], [14], and conversational approaches have explored interactive hardware design [15]. Power, performance, and area optimization-aware Verilog generation with large language models represents a particularly relevant direction because it explicitly connects generated code to synthesis objectives rather than treating code generation as an end in itself [10].

## 3. LLM-Enabled High-Level Synthesis Architecture

A credible LLM-driven high-level synthesis architecture should separate generative flexibility from verification authority. In such a system, the language model acts as an interface agent that translates design intent into candidate high-level synthesis specifications, directive sets, and hardware description language modules. The synthesis tool remains the source of truth for resource estimates, timing, and power. The verification stack remains the source of truth for functional correctness and safety properties. This separation is not merely technical; it is a governance mechanism that prevents plausible text from being mistaken for verified silicon. The model may propose, explain, and refactor, but it should not be allowed to quietly redefine constraints or bypass checks.

The architecture typically includes several interacting layers. The intent layer captures natural-language requirements, constraints, and design rationale. The representation layer converts those requirements into structured artifacts such as behavioral code, pragma configurations, interface specifications, and testbench hypotheses. The synthesis layer invokes conventional high-level synthesis and logic synthesis tools, returning reports that the model can interpret. The verification layer includes simulation, linting, equivalence checking, and property-based verification where applicable. The learning layer stores successful transformations, failure modes, and human feedback so that future proposals improve without erasing provenance. Finally, the governance layer enforces access control, audit logging, licensing review, and approval gates for safety-critical changes. This layered view treats LLMs as components in a socio-technical infrastructure rather than as standalone design authorities [16].

The interaction model matters as much as the layer structure. A purely autonomous agent that repeatedly generates and synthesizes code may produce impressive demonstrations but poor accountability. A human-in-the-loop system can assign different levels of autonomy based on risk. For example, the model might freely suggest comments, testbenches, or documentation, but require human approval before changing clock-domain crossing logic, memory

consistency assumptions, or power domains. The model might also produce multiple candidate designs and explanations, leaving the engineer to select among trade-offs. This approach aligns with human-AI interaction principles that emphasize clear scope, feedback, and user control [20]. In high-level synthesis, such control is essential because design intent often includes tacit constraints that are not written in the source code.

## 4. Structural Trade-offs in Model, Compiler, and Hardware Co-Design

The introduction of language models into high-level synthesis creates structural trade-offs that cannot be resolved by model scaling alone. The first trade-off is between abstraction and predictability. Higher abstraction allows faster exploration and broader participation, but it also hides implementation details that determine timing and power. A model that generates concise C++ may obscure memory access patterns that later cause synthesis failures. Conversely, a model that generates highly explicit hardware description may improve predictability but eliminate the productivity benefits of high-level synthesis. The optimal balance depends on design maturity, team expertise, and the cost of late-stage changes.

The second trade-off is between resource efficiency and verifiability. Aggressive optimization may involve speculative transformations, approximate computing, or unusual pipeline structures that are difficult to verify. Conservative transformations may be easier to check but leave performance on the table. LLMs can propose both, but they cannot independently establish that a transformation preserves semantics. The system must therefore allocate verification effort according to risk. Safety-critical blocks, security-sensitive interfaces, and analog-mixed-signal boundaries require stronger evidence than non-critical accelerators or test infrastructure. This suggests a tiered autonomy model in which model-generated changes are accepted only when their verification burden is commensurate with their expected benefit.

The third trade-off is between toolchain integration and model generality. A general-purpose language model can operate across many languages and tools, but deep optimization requires tool-specific knowledge. A domain-adapted model may perform better on a particular synthesis flow, but it may overfit to one vendor, one process node, or one benchmark family [13]. The architecture should therefore separate reusable reasoning about design patterns from tool-specific adapters that encode directives, report formats, and constraint syntax. This separation reduces lock-in and allows the organization to update the model as tools and technology libraries evolve. It also makes evaluation more transparent because improvements can be attributed to reasoning, adaptation, or tool integration rather than to an opaque combination.

## 5. Data, Verification, and Robustness Infrastructure

Data infrastructure is a first-order concern for LLM-driven high-level synthesis. Hardware design data is scarce relative to software code, often proprietary, and frequently protected by non-disclosure agreements. Public benchmarks are useful for research but may not represent industrial complexity, legacy interfaces, or process-specific constraints [14]. Training on proprietary data raises questions about licensing, provenance, and contamination. Even when data is available, labels are expensive because they require synthesis runs, simulation, formal checks, and expert judgment. The system must therefore support data-efficient learning, retrieval from curated internal repositories, and continuous evaluation against held-out designs that reflect real deployment conditions.

Verification infrastructure must be robust to the specific failure modes of language models. These include syntactic errors, semantic hallucinations, unsafe assumptions about timing,

incorrect handling of reset and initialization, and subtle deviations from interface protocols. Simulation can catch many functional errors, but simulation coverage is itself a design problem. Formal verification can provide stronger guarantees, but it depends on the availability of specifications and properties. Equivalence checking can compare generated hardware against a reference implementation, but it requires a trusted reference. A layered verification strategy should combine linting, simulation, coverage analysis, formal property checking, equivalence checking, and post-synthesis timing analysis. The model can help generate testbenches and properties, but its outputs must be validated by independent tools.

Robustness also concerns distribution shift. A model trained on academic Verilog may perform poorly on industrial SystemVerilog with complex interfaces. A model tuned for FPGA flows may propose directives that are invalid for ASIC standard-cell libraries. A model exposed to new process nodes may lack knowledge of updated design rules. The system should detect when inputs fall outside the validated distribution and fall back to conservative behavior or human review. It should also resist adversarial prompts that attempt to bypass design rules or insert malicious logic. Security in hardware design is not only about confidentiality; it includes integrity, supply-chain trust, and protection against trojan insertion. LLM-driven optimization must therefore be evaluated as part of a secure development lifecycle, not only as a productivity tool.

## 6. Governance, Fairness, and Sustainability

Governance is essential because LLM-driven high-level synthesis redistributes expertise and authority. If a model proposes a design and a junior engineer accepts it, responsibility for defects may become ambiguous. Organizations need policies that define who is accountable for model-assisted decisions, how model outputs are recorded, and how disputes are resolved. Audit trails should capture prompts, model versions, retrieved documents, synthesis reports, verification results, and human approvals. These records support reproducibility, incident analysis, and regulatory review. They also help organizations learn from failures without relying on memory or informal communication.

Fairness and access are equally important. High-quality language models, synthesis tools, and compute infrastructure are expensive. If only large firms and well-funded universities can use advanced LLM-driven optimization, the benefits may concentrate while smaller innovators face higher barriers. Open benchmarks, shared verification suites, and transparent evaluation protocols can partially mitigate this imbalance [14]. However, openness must be balanced against security and intellectual property concerns. A fair ecosystem would provide tiered access to models and data, support reproducibility through standardized reporting, and avoid evaluation practices that reward benchmark overfitting rather than genuine design improvement. Ethical analysis of algorithms reminds us that fairness is not only about allocation of resources but also about the power to define what counts as a good design [17].

Sustainability is another dimension. Repeated LLM inference, especially with large models and long design iterations, consumes energy and compute capacity [18], [19]. The environmental cost may be justified if it reduces expensive silicon respins or improves energy efficiency at deployment. But if the model is used for trivial tasks or generates many low-quality candidates, the net benefit may be negative. Sustainable deployment requires caching, retrieval, smaller task-specific models, and early filtering of implausible proposals. It also requires measuring the full lifecycle impact, including training, inference, synthesis runs, and verification. A design flow that saves one mask revision may justify substantial computation; a flow that merely shifts effort from engineers to servers may not.

## 7. Deployment and Operational Considerations

Deployment of LLM-driven high-level synthesis in an industrial setting involves more than model selection. It requires integration with version control, continuous integration, artifact repositories, and electronic design automation schedulers. The model must respect access controls and not leak proprietary designs through prompts or logs. It must handle long-running synthesis jobs asynchronously and provide useful summaries without overwhelming engineers. It must degrade gracefully when tools fail, licenses expire, or compute quotas are exhausted. These operational concerns are similar to those studied in machine learning systems, where hidden technical debt arises from entanglement, undeclared consumers, and configuration complexity [16].

The deployment model also affects latency and trust. A cloud-hosted model may provide high capability but raises data residency and confidentiality issues. A locally hosted model may improve privacy and latency but may have lower quality or higher maintenance cost. A hybrid approach might use local models for routine transformations and cloud models for difficult reasoning, with strict controls on what data leaves the premises. The choice depends on the sensitivity of the design, the regulatory environment, and the organization's tolerance for operational risk. In ASIC development, where design data can represent years of investment, confidentiality is often non-negotiable.

Operational metrics should go beyond functional correctness. They should include synthesis quality, verification coverage, time to closure, number of human interventions, and rate of rejected proposals. They should also include cost, energy, and fairness indicators, such as whether certain teams are systematically disadvantaged by model availability. A deployment that improves average productivity but creates a new bottleneck in verification may not be successful. A deployment that generates many candidates but few acceptable ones may increase technical debt. The most useful systems will be those that improve the overall design process while preserving accountability and traceability.

## 8. Evaluation, Case Illustrations, and Cross-Domain Comparisons

Evaluation of LLM-driven high-level synthesis should be scenario-based rather than solely benchmark-based. Small benchmark problems can measure syntax generation and functional correctness, but they rarely capture the complexity of memory hierarchies, clock-domain crossings, power domains, or physical constraints. A stronger evaluation suite would include progressively harder tasks: generating synthesizable modules from specifications, optimizing directive sets for known designs, repairing synthesis errors, refactoring code for better timing, and exploring architecture alternatives under fixed area budgets. Each task should report not only success rates but also verification cost, synthesis runtime, and human review effort. This multidimensional view prevents misleading conclusions based on narrow metrics.

Case illustrations from different domains clarify the trade-offs. In a cryptographic accelerator, the model might suggest parallelizing modular multiplication, but formal verification and side-channel analysis are essential because performance gains cannot justify security weaknesses. In an image signal processor, the model might propose memory tiling and line-buffer reuse, but the evaluation must consider bandwidth, latency, and quality-of-service guarantees. In a machine learning accelerator, the model might generate dataflow schedules, but the design must remain robust across model architectures and compiler versions. In each case, the LLM is most useful when it generates alternatives and explanations that engineers can test, not when it makes irreversible decisions.

Cross-domain comparison with software engineering shows both promise and risk. LLM code assistants can improve developer productivity, but they also introduce insecure code, license issues, and overreliance. Hardware design amplifies these risks because errors are costly and physical. Unlike software, hardware cannot always be patched after deployment. Unlike FPGA designs, ASIC designs cannot be reconfigured. This asymmetry means that LLM-driven ASIC optimization should adopt stronger verification and governance than typical software assistants. The comparison also suggests that human-AI guidelines developed for other domains can inform hardware design, especially regarding transparency, user control, and error recovery [20]. The goal is not to eliminate human expertise but to amplify it while keeping final authority with accountable engineers.

## 9. Policy and Future Directions

Policy for LLM-driven high-level synthesis should address safety, security, competition, and environmental impact. Safety-critical silicon used in medical devices, automotive systems, industrial control, and aerospace may require documented processes for model-assisted design. Regulators may not need to approve every model, but they may need to ensure that organizations can explain how generated artifacts were verified and who approved them. Security policy should address model supply chains, data leakage, and adversarial manipulation. Competition policy should consider whether access to advanced models and synthesis tools becomes a barrier to entry. Environmental policy should encourage measurement and reporting of compute and energy use in design flows [18], [19].

Future research should pursue several directions. First, methods for grounded generation that tightly couple language models to synthesis and verification feedback. Second, verification-aware optimization that treats proof obligations as part of the objective, not an afterthought. Third, data-efficient learning that respects intellectual property while enabling domain adaptation. Fourth, human-AI interaction models that allocate autonomy according to risk and expertise. Fifth, evaluation frameworks that measure system-level outcomes rather than isolated code correctness. Sixth, governance mechanisms that provide accountability without stifling innovation. These directions are interdependent; progress in one area without the others may produce brittle or inequitable systems.

## 10. Conclusion

LLM-driven high-level synthesis optimization for application-specific integrated circuits is a promising but demanding research area. Language models can reduce the friction between human intent and hardware implementation by generating candidate designs, explaining synthesis reports, and exploring directive spaces. However, their outputs are probabilistic and unverified, while ASIC design is unforgiving of error. The only credible path forward is a system-level approach that integrates generative models with synthesis tools, formal and simulation-based verification, data governance, human oversight, and responsible deployment practices. The architecture should preserve clear boundaries between proposal and authority, allowing models to suggest while engineers and verification tools decide.

The structural trade-offs among abstraction, predictability, resource efficiency, verification burden, and accountability must be managed explicitly. Robustness requires attention to distribution shift, adversarial manipulation, and supply-chain security. Fairness requires equitable access to models, data, and evaluation infrastructure. Sustainability requires measuring the energy and compute cost of iterative inference against the benefits of better silicon. Policy should support safety and security without locking out smaller innovators. If

these conditions are met, LLM-driven high-level synthesis can become a valuable part of the electronic design automation ecosystem, not as an autonomous designer but as an auditable collaborator in the difficult work of turning intent into efficient and trustworthy silicon.

**References**


1. Gajski, D. D., Dutt, N. D., Wu, A. C. J., & Lin, S. Y. (1992). High-level synthesis: Introduction to chip and system design. Springer.

2. Micheli, G. (1994). Synthesis and optimization of digital circuits. McGraw-Hill.

3. Coussy, P., Gajski, D. D., Meredith, M., & Takach, A. (2009). An introduction to high-level synthesis. IEEE Design & Test of Computers, 26(4), 8–17.

4. Nane, R., Sima, V. M., Pilato, C., Choi, J., Fort, B., Canis, A., ... & Bertels, K. (2016). A survey and evaluation of FPGA high-level synthesis tools. IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems, 35(10), 1591–1604.

5. Canis, A., Choi, J., Aldham, M., Zhang, V., Kammoona, A., Anderson, J. H., Brown, S., & Czajkowski, T. (2011). LegUp: High-level synthesis for FPGA-based processor/accelerator systems. In Proceedings of the 19th ACM/SIGDA International Symposium on Field Programmable Gate Arrays (pp. 33–36). ACM.

6. Cong, J., Lau, J., Liu, G., Neuendorffer, S., Pan, P., Vissers, K., & Zhang, Z. (2022). FPGA HLS today: Successes, challenges, and opportunities. ACM Transactions on Reconfigurable Technology and Systems, 15(4), 1–42.

7. Bacon, D. F., Graham, S. L., & Sharp, O. J. (1994). Compiler transformations for high-performance computing. ACM Computing Surveys, 26(4), 345–420.

8. Teich, J. (2012). Hardware/software codesign: The past, present, and future. Proceedings of the IEEE, 100(Special Centennial Issue), 1417–1430.

9. Lee, E. A. (2008). Cyber physical systems: Design challenges. In 2008 11th IEEE International Symposium on Object and Component-Oriented Real-Time Distributed Computing (ISORC) (pp. 363–369). IEEE.

10. Thorat, K., Zhao, J., Liu, Y., Hasan, A., Peng, H., Xie, X., ... & Ding, C. (2025, September). LLM-VeriPPA: Power, Performance, and Area Optimization aware Verilog Code Generation with Large Language Models. In 2025 ACM/IEEE 7th Symposium on Machine Learning for CAD (MLCAD) (pp. 1-7). IEEE.

11. Pearce, H., Tan, B., & Karri, R. (2020). DAVE: Deriving automatically Verilog from English. In Proceedings of the 2020 ACM/IEEE Workshop on Machine Learning for CAD (pp. 27–32). ACM.

12. Thakur, S., Ahmad, B., Pearce, H., Tan, B., Dolan-Gavitt, B., Karri, R., & Garg, S. (2024). VeriGen: A large language model for Verilog code generation. ACM Transactions on Design Automation of Electronic Systems, 29(3), 1–31.

13. Liu, M., Ene, T., Xu, M., et al. (2023). ChipNeMo: Domain-adapted LLMs for chip design. arXiv preprint arXiv:2311.00176.

14. Lu, Y., Liu, S., Zhang, Q., & Xie, Z. (2024). RTLLM: An open-source benchmark for design RTL generation with large language model. In 2024 29th Asia and South Pacific Design Automation Conference (ASP-DAC) (pp. 722–727). IEEE.

15. Blocklove, J., Garg, S., Karri, R., & Pearce, H. (2023). Chip-Chat: Challenges and opportunities in conversational hardware design. In 2023 ACM/IEEE 5th Workshop on Machine Learning for CAD (MLCAD) (pp. 1–6). IEEE.

16. Sculley, D., Holt, G., Golovin, D., Davydov, E., Phillips, T., Ebner, D., ... & Dennison, D. (2015). Hidden technical debt in machine learning systems. In Advances in Neural Information Processing Systems 28 (pp. 2503–2511). Curran Associates.

17. Mittelstadt, B. D., Allo, P., Taddeo, M., Wachter, S., & Floridi, L. (2016). The ethics of algorithms: Mapping the debate. Big Data & Society, 3(2), 1–21.

18. Strubell, E., Ganesh, A., & McCallum, A. (2019). Energy and policy considerations for deep learning in NLP. In Proceedings of the 57th Annual Meeting of the Association for Computational Linguistics (pp. 3645–3650). ACL.

19. Patterson, D., Gonzalez, J., Le, Q., Liang, C., Munguia, L.-M., Rothchild, D., So, D., Texier, M., & Dean, J. (2021). Carbon emissions and large neural network research. arXiv preprint arXiv:2104.10350.

20. Amershi, S., Weld, D., Vorvoreanu, M., Fourney, A., Nushi, B., Collisson, P., ... & Horvitz, E. (2019). Guidelines for human-AI interaction. In Proceedings of the 2019 CHI Conference on Human Factors in Computing Systems (pp. 1–13). ACM.