

# Focus Session: Large Language Models in Physical Design: From Data Generation to Intelligent Agents

Bing-Yue Wu<sup>\*1</sup>, Atmadip Dey<sup>\*1</sup>, Austin Rovinski<sup>2</sup>, and Vidya A. Chhabria<sup>1</sup>  
<sup>1</sup>Arizona State University; <sup>2</sup>New York University

**Abstract**—Physical design remains one of the most complex stages of chip implementation, requiring deep expertise in electronic design automation (EDA) tools, workflows, and design knowledge. While open-source EDA tools have improved accessibility and reproducibility, the effective use of physical design flows still requires significant manual effort and domain expertise. In parallel, large language models (LLMs) have rapidly evolved from data-driven language models to assistants and tool-interacting agents capable of reasoning, code generation, and closed-loop execution. This paper presents a perspective on the evolution of LLM usage in physical design, tracing a progression from early data-driven question answering and script generation to tool-aware assistants and closed-loop agentic workflows for physical design tasks. This paper highlights this evolution using representative open-source efforts and case studies. Further, we outline emerging research directions in which agentic LLMs move toward optimization and algorithm discovery, including the automation of tasks such as engineering change orders (ECOs) and the exploration of algorithm discovery within physical design tools. The work also highlights opportunities and challenges of LLM-driven design automation.

## I. INTRODUCTION

Physical design is a complex and expertise-intensive stage of chip implementation. Modern electronic design automation (EDA) tools for physical design expose hundreds of commands, configuration options, and multi-stage workflows spanning placement, routing, clock-tree synthesis, timing optimization, and sign-off, as shown in Fig. 1. Effective use of these tools requires substantial domain experience, and even small changes in tool versions or design constraints can have nonintuitive effects on the quality of results (QoR) and runtime. As designs continue to grow in scale, this complexity increasingly limits productivity and accessibility.

Open-source EDA tools and flows, such as OpenROAD [1]–[3] and OpenROAD-flow-scripts [4], have emerged as an important response to these challenges by improving accessibility and enabling standardized, push-button physical design flows. By providing carefully selected default settings and end-to-end automation, these flows reduce users’ cognitive burden by limiting unnecessary configuration choices, enabling non-expert users to obtain reasonable results without extensive tool knowledge [1], [2]. This design philosophy has enabled broader participation from academia, startups, and students and lowered barriers to experimentation through reproducible workflows and standardized benchmarks. However, while open-source tools address access and licensing barriers, they do not eliminate the need for expertise, particularly when

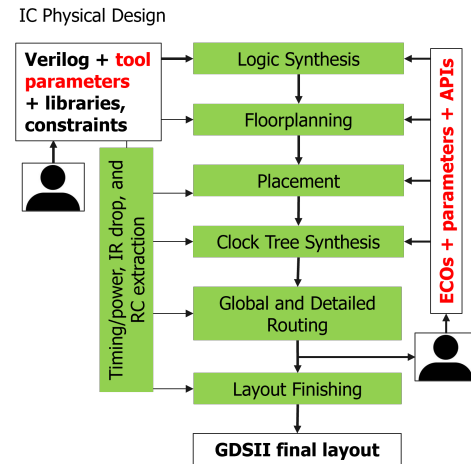


Fig. 1. Complexity of the physical design flow with manually-provided tool parameters, ECO commands, and iterations across stages.

to tape-out chips. Design workflows remain intricate, documentation of open-source tools is often incomplete, and tool interfaces and APIs continue to evolve. Therefore, significant portions of the design process still rely on manual intervention and expert knowledge.

In parallel, large language models (LLMs) have undergone rapid evolution in the broader artificial intelligence (AI) community. Early efforts focused on curating large-scale datasets [5] and finetuning general-purpose language models [6] to support question answering [7], code generation [8], and interactive assistants [9]. More recently, LLMs have been integrated into agentic systems that interact directly with external tools [10], [11], execute actions, observe outcomes, and iteratively refine behavior through feedback. Advances in reasoning and planning have further expanded the role of LLMs toward solving complex problems.

These developments in LLMs have begun to influence physical design. LLM usage has evolved in stages, mirroring trends observed in other domains. Initial efforts focused on data collection [12], supervised fine tuning of existing foundation models, and question answering to improve understanding of tool usage and workflows. This was followed by LLM-based assistants capable of generating scripts and responding to tool-specific queries [13], [14]. More recent work has explored closed-loop, tool-interacting agents that can execute, debug, and refine physical design workflows [15], [16]. Recently, agents for algorithmic discovery have also been adopted to address the satisfiability problem [17] and OpenROAD quality

\* denotes equal contribution

of results (QoR) [18]. The work in [19] establishes a fundamental distinction between agents that interact with existing EDA tools in a flow such as [15], [16] and those that generate EDA tool source code, such as [17], [18], naming them *flow-level* and *code-level* agents, respectively.

This paper provides a structured view of the evolution of LLMs in physical design, grounded in representative case studies of open-source efforts. The paper highlights challenges in adopting LLMs and discusses emerging opportunities, including automation of engineering change orders (ECOs), deep reasoning-based optimization, and algorithm discovery. The paper is organized as follows. Section II provides background on physical design workflows, open-source EDA, and AI in EDA. Section III examines the early and advanced stages of LLM adoption in physical design. Section IV discusses key challenges and implications. Section V concludes the paper.

## II. BACKGROUND

### A. Physical design flow complexity

Physical design involves tightly coupled optimization problems such as placement, clock tree synthesis, and routing, solved using heuristic-driven EDA tools. Although automated, these tools depend heavily on user constraints and tuning, and early decisions can significantly impact QoR. Since each design differs, parameter settings rarely generalize, leading to repeated iterations with a human in the loop, as highlighted in Fig. 1. As a result, physical design remains human-in-the-loop: designers adjust parameters, diagnose violations, and apply ECOs, relying on global reasoning beyond the tools' local optimizations.

### B. AI in physical design

To address this complexity, AI has long been explored in physical design, with its adoption closely following the broader evolution of AI. Fig. 2 (top) shows a timeline of its evolution. Early efforts in the mid-2010s focused on regression-based models, such as support vector machines (SVMs) and random forests, for predictive tasks including timing analysis [20], lithography hotspot detection [21], and congestion prediction [22]. This was followed by the deep learning era, in which convolutional neural networks (CNNs) and U-Net architectures enabled image-based representations of layouts, enabling accurate prediction of physical effects such as congestion [23], power integrity [24], voltage drop [25], and design-rule violations [26] as fast surrogates.

Subsequently, the field shifted from prediction to optimization, leveraging graph neural networks and reinforcement learning for problems such as gate sizing [27], [28], floorplanning [29], and EDA tool parameter tuning [30], [31]. More recently, generative AI-enabled models capable of producing scripts [14], generating Verilog [32]–[34], and natural-language interfaces to design tools [13], [14], [35] have emerged. This progression culminates in the emerging agentic AI era, in which LLM-based agents can read design reports, execute tool commands, analyze outcomes, and iteratively refine solutions—tasks that previously required substantial manual effort [15], [16]. Together, these developments reflect

a steady shift of AI toward the core decision-making loop. Newer AI-enabled systems in commercial EDA have also emerged [30], [36].

### C. Role of open-source in AI-driven physical design

Open-source EDA plays a central role in enabling the AI revolution in physical design by offering a best-of-both-worlds paradigm: accessibility and automation for non-experts, and maximum flexibility for experts [37]. Open-source tools allow users to inspect, modify, and extend algorithms and parameters, which is critical where no single strategy generalizes across designs. This flexibility enables rapid experimentation, custom optimization objectives, and design-specific tuning, while still supporting reproducible, script-driven workflows.

Equally important, open-source infrastructure enables friction-free integration with modern AI systems. Open tool code, APIs, PDKs, datasets, and benchmarks expose commands, logs, and intermediate artifacts directly to online LLMs and agentic systems without legal constraints. This enables closed-loop, agent-driven physical design workflows that reason over tool behavior rather than abstractions. In this emerging ecosystem, open-source and commercial EDA coexist [38]. Commercial tools provide production robustness, while open-source platforms offer transparency, flexibility, and AI readiness for LLM-based assistants, automated parameter discovery, and algorithm discovery.

## III. EVOLUTION OF LLM USAGE IN PHYSICAL DESIGN

The evolution of LLM usage in physical design has largely mirrored the trajectory of LLM research in the AI community. Several key advances in LLM architectures, training paradigms, and deployment models have had a direct, outsized impact on the adoption of these models within physical design. Fig. 2 illustrates this co-evolution: the lower half of the timeline in the red inset highlights major milestones in general LLM research, while the upper half captures corresponding advances in the application of LLMs to physical design. The figure is color-coded by major LLM research themes (finetuning, chatbots, code generation, agentic workflows) to emphasize how developments in the broader AI ecosystem map onto physical design use cases.

**Key LLM advancements.** Early LLMs were primarily conversational systems trained for next-token prediction, capable of dialogue and question answering but lacking reliability, controllability, and task specificity [7].

A major shift occurred with the emergence of code generation. In 2021, Codex [10] demonstrated that scaling LLMs on large code corpora enabled the synthesis of executable programs from natural language, transforming LLMs from conversational agents into productive tools. In 2022, subsequent advances addressed the limitations of one-shot generation. AlphaCode [8] reframed code generation as a search problem through large-scale sampling and automated evaluation, while instruction tuning and reinforcement learning from human feedback (RLHF) [39] enabled models to better follow user intent and constraints. In parallel, parameter-efficient adaptation techniques such as LoRA [40] made it computationally

|                    | Regression era   | Deep learning era   | RL and graph era  | Generative AI era  | Agentic AI era   |
|--------------------|--|---|---|--|--|
|                    | 2015 – 2018  | 2018-2020   | 2020-2022   | 2022-2024  | 2024-present   |
| <b>Models</b>      | SVMs, RFs, XGBoost.  | CNNs, Unets, MLP, RNNs.   | GCN and deep RL optimization  | Transformers: LLMs   | LLMs, multi-modal AI models  |
| <b>Application</b> | Timing analysis, lithography hotspot detection, congestion hotspots  | Layout congestion, IR drop, DRC prediction  | Logic gate sizing, floorplanning, parameter tuning  | Chatbots, script generators, Verilog generation  | Hyper-parameter tuning, EDA tool code generation   |
| <b>Examples</b>    | 2017, Lithography and DRC hotspot Yang, et al. [21]<br>2020, Power grids Chhabria et al. [24]<br>2015, Timing analysis Kahng et al. [20] | 2021, IR drop prediction Chhabria, et al. [25]<br>2020, Power grids Chhabria et al. [24]<br>2018, Congestion prediction Xie et al. [23] | 2022, 2023, Gate sizing Lu, et al. [27], Jiang, et al. [28]<br>2026, Parameter tuning Synopsys DSO.ai [30]<br>Cadence Cerebrus [31] | 2023, 2024, Chatbots Wu, et al. [14], Wu, et al. [13], Wu, et al. [16]<br>2024, Verilog generation Liu et al. [34], Thakur et al. [32] | 2025, Design flow Ghose et al. [15], Wu et al. [16]<br>2025, Verilog correction Chang et al. [33]<br>2026, EDA tool source code generation Ghose et al. [18] |

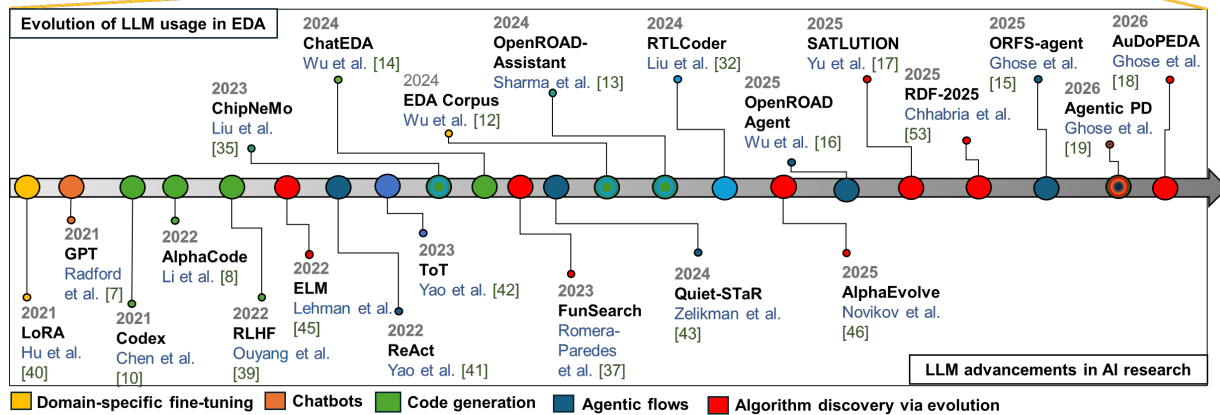


Fig. 2. Evolution of AI in physical design with emphasis on LLM usage, contrasted with major advances in LLM research.

practical to specialize large foundation models for narrow domains without retraining them from scratch. This capability was critical to the proliferation of domain-specific LLMs while preserving the benefits of large-scale pretraining.

From late 2022 onward, research increasingly emphasized reasoning, interaction, and agency. Frameworks such as ReAct [41], Tree of Thoughts (ToT) [42], and Quiet-STaR [43] recast inference as a multi-step decision process involving planning, tool use, and iterative refinement, rather than a single-pass response. This reframing laid the groundwork for treating LLMs not merely as generators, but as decision-making systems operating within closed-loop workflows. Next, LLMs began to play an explicit role in optimization and discovery. Early efforts such as FunSearch [44] and ELM [45] demonstrated that LLMs could serve as generative operators within evolutionary loops. This idea was advanced by AlphaEvolve [46], which integrates LLMs into large-scale evolutionary optimization pipelines that operate over entire codebases and algorithmic systems. By coupling LLM-driven proposal generation with automated evaluation, selection, and iteration, AlphaEvolve marks a transition from LLMs as interactive assistants to LLMs as autonomous components. These

advances trace a progression—from conversational models, to task-oriented assistants, to agentic systems embedded within closed-loop optimization frameworks and scientific discovery. This sets the stage for their adoption in domains such as physical design, where iterative reasoning, tool interaction, and objective-driven optimization are fundamental requirements.

**LLM advancements in physical design.** In the remainder of this section, we highlight concrete examples drawn primarily from OpenROAD-related [1] efforts that illustrate the evolution of LLM applications in physical design, spanning early assistant-style interactions, agent-based integration with design tools (flow-level), and emerging directions in optimization and algorithm discovery.

#### A. Datasets as the first bottleneck

In early 2023, general-purpose LLMs available online performed poorly on physical design tasks, largely due to the scarcity of open and domain-relevant training data. This limitation motivated early efforts to curate physical design datasets and to fine-tune domain-specific models tailored to EDA workflows. Efforts such as the EDA Corpus [12] and ORD-QA [47] created datasets for OpenROAD.

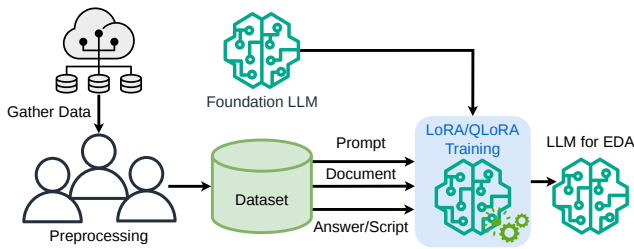


Fig. 3. Supervised finetuning flow to create domain-adapted LLMs.

EDA Corpus [12] is an open-source, expert-curated dataset created to address two practical bottlenecks in the use of real-world physical design tools. The first is tool question answering, where users repeatedly face installation issues and questions about the underlying EDA tools and APIs, and where limited documentation often forces a slow loop of searching manuals, reading scattered examples, and inspecting source code. The second is tool script generation, where a user’s natural-language intent must be translated into executable scripts that correctly invoke tool APIs. It pairs realistic user prompts with expert-verified explanations and tool-validated OpenROAD scripts, enabling LLMs to both explain tool functionality and generate executable physical design scripts. It consists of 198 unique question–answer pairs, expanded to nearly 600 through paraphrasing. It also includes 396 prompt–script pairs covering physical design stages, as well as OpenDB-based queries and edits, and is further augmented to nearly 1,000 pairs.

ORD-QA [47] is a benchmark for evaluating QA systems on OpenROAD documentation. It contains questions with their corresponding retrieved document and reference answers, covering command usage, parameters, and design flow procedures. Performance on this benchmark is evaluated by comparing generated responses with ground-truth answers using accuracy and answer quality metrics.

### B. LLMs as a chatbot for physical design

Building on the aforementioned datasets, researchers have trained existing foundation models to build domain-specific chatbots and assistants. A few example assistants include [13], [14], [47]–[49]. These approaches follow a typical approach of supervised fine tuning (SFT) using a curated dataset. This flow for finetuning existing models is shown in Fig 3.

OpenROAD-Assistant [13] is trained to respond to OpenROAD-related queries, including tool-related questions and the generation of OpenROAD Python scripts using OpenROAD APIs. It fine-tunes a Llama3 foundation using QLoRA [40]. It further adopts retrieval-augmented finetuning (RAFT) [50], which integrates supervised learning with retrieved context from the OpenROAD API documentation and tool manuals during training. This approach improves performance and mitigates hallucinations when the model is later integrated with a retrieval-augmented generation (RAG) system. Although standard RAG augments prompts with external information, it may introduce errors due to irrelevant

or incorrect retrieval. In contrast, RAFT trains the model to extract relevant information from context-rich prompts, leading to more accurate and reliable responses.

ChatEDA [14] proposes an LLM-powered agent to automate physical design workflows. It integrates an expert LLM with OpenROAD and iEDA [51] to enable natural-language-driven tool execution. ChatEDA interprets user intent, decomposes complex objectives into subtasks, generates high-level physical design flow scripts, and invokes tools while refining actions based on feedback. A translator module bridges LLM-generated scripts and actual tool-specific commands to ensure correct execution.

RAG-EDA [47] proposes a customized RAG pipeline for answering questions about OpenROAD documentation. It includes three fine-tuned modules: (1) a retriever for accurate document retrieval, (2) a reranker to refine candidate ranking, and (3) an LLM adapted through domain-specific pretraining and instruction finetuning to better understand EDA terminology. The system addresses questions related to command usage, parameter settings, and design flow steps.

ORAssistant [48] proposes a RAG-based assistant for the OpenROAD EDA flow. It enhances the OpenROAD user experience by answering questions related to installation, command usage, flow configuration, execution, and debugging. The system builds a RAG database from OpenROAD documentation and GitHub resources. ORAssistant employs a hybrid retrieval strategy that combines semantic and keyword search to retrieve relevant documents, which are then provided to a base LLM to generate grounded responses. ORAssistant demonstrates improved accuracy over standalone LLMs and is designed to scale to other open-source EDA tools.

### C. LLMs as closed loop agents for physical design automation

The generated one-time outputs from chatbots and assistants still require manual validation. Physical design work is inherently interactive, and productivity losses often stem from repeated cycles of locating commands, interpreting tool behavior, and debugging API usage as one moves across flow stages or adopts a new tool. This motivates the development of agents that rely on concepts such as ReAct [41] and ToT [42] where a foundation model runs an EDA tool and uses logs, metrics and data generated by the tool to generate its output. This process is performed iteratively until the task in the prompt is completed, as shown in Fig. 4.

Examples in physical design include OpenROAD Agent [16]. It extends the chatbot-based OpenROAD-Assistant into an execution-grounded framework that allows the model to interact directly with the OpenROAD tool. During inference, OpenROAD Agent generates a script from a user inquiry, runs it in OpenROAD through a Python subprocess, captures real-time feedback such as warnings and error messages caused by hallucinated API calls or incorrect argument passing, and then iteratively refines the script using tool feedback until the generated script executes without errors. In OpenROAD Agent, this loop focuses on two common issues: calling the wrong API and passing the wrong

TABLE I  
EVALUATION OF OPENROAD AGENT: COMPARISON AGAINST FOUNDATION MODELS, OPENROAD ASSISTANT, AND ABLATIONS [16].

| Category         | Models and training           | Pass@K (noniterative) |        | Iterative   |       |            |
|------------------|-------------------------------|-----------------------|--------|-------------|-------|------------|
|                  |                               | Pass@1                | Pass@3 | avg. #iter. | Pass% | Pass cases |
| OpenROAD Agent   | 32B (1st + 2nd + 3rd stage)   | 80.00%                | 81.49% | 1.19        | 94%   | 131        |
|                  | 7B (1st + 2nd + 3rd stage)    | 70.00%                | 79.29% | 1.11        | 92%   | 129        |
| Ablation Studies | 32B (1st + 2nd stage)         | 81.49%                | 82.86% | 1.15        | 93%   | 130        |
|                  | 32B (1st stage)               | 80.00%                | 81.43% | 1.13        | 92%   | 129        |
|                  | 7B (1st + 2nd stage)          | 79.29%                | 83.57% | 1.13        | 91%   | 127        |
|                  | 7B (1st stage)                | 77.14%                | 82.86% | 1.11        | 91%   | 128        |
| Baselines        | OpenROAD-Assistant retrained* | 73.57%                | 75.00% | 1.04        | 78%   | 109        |
|                  | OpenROAD-Assistant*           | 9.29%                 | 13.57% | 1.13        | 14%   | 19         |
|                  | Qwen2.5-Coder-32B-Instruct    | 3.57%                 | 5.00%  | 2.15        | 14%   | 20         |
|                  | Qwen2.5-Coder-7B-Instruct     | 0.00%                 | 0.00%  | 0.00        | 0%    | 0          |
|                  | ChatGPT-4o-mini               | 0.00%                 | 0.00%  | 4.00        | 1%    | 1          |
|                  | Llama-3.3-70B-Instruct        | 0.00%                 | 0.00%  | 0.00        | 0%    | 0          |

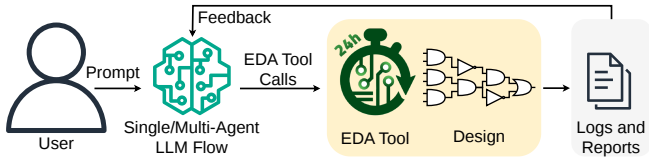


Fig. 4. Framework for agentic interaction between EDA tool and LLMs.

arguments. By repeatedly using tool feedback to revise the script, the agent becomes less likely to produce API-related hallucinations that static generation cannot readily avoid. Table I compares performance across the training stages of OpenROAD Agent based on two model sizes from the Qwen series. The results show that the three-stage training scheme is effective for both model sizes and substantially outperforms all baselines.

#### D. LLMs for agentic optimization in physical design

Optimization in physical design is a complex process, and several optimization tasks that improve result quality remain highly manual today. These include selecting tool parameters, performing engineering change orders (ECOs), and achieving design closure through last-mile optimizations such as fixing timing and design rule violations. As illustrated by the tightly coupled stages in the physical design flow in Fig. 1, these tasks are inherently iterative, context-dependent, and difficult to capture with fixed heuristics. Recent work has explored the use of LLMs as agents that operate in closed-loop interaction with EDA tools. By observing intermediate tool outputs, reasoning over design context, and iteratively proposing targeted changes, LLM-based agents offer a promising alternative to both manual tuning and purely black-box optimization. Tasks such as parameter tuning, timing ECOs, and DRC fixing are particularly well-suited to this paradigm, as they require sequential decision-making informed by both local metrics and global design objectives. The techniques are classified by [19] as flow-level agents.

##### 1) Agents for flow parameter selection

ORFS-agent [15] is an example of LLM-driven agentic optimization applied to physical design. The system targets a concrete and measurable task within the OpenROAD flow: selecting tool hyperparameters, such as core utilization and

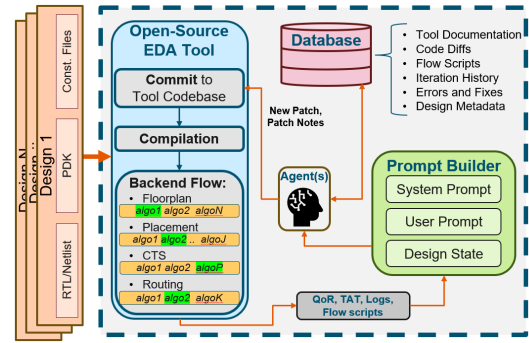


Fig. 5. Design-tool co-evolution flow to enable a custom EDA tool (algorithm or source code) tailored to a specific design.

constraint settings, that directly influence final QoR. Rather than treating tuning as a blind search problem, ORFS-agent places an LLM in the optimization loop to reason about the effects of prior runs and propose improved configurations.

The agent follows an iterative loop of observe, analyze, propose, and apply. It launches batches of OpenROAD runs, aggregates configuration choices and resulting metrics, and then reasons over this structured context to generate the next set of candidate parameters. Across multiple PDKs, ORFS-agent demonstrates improvements in key QoR metrics while requiring fewer iterations than traditional autotuning approaches. The framework is modular and model-agnostic, allowing different LLMs or optimization backends to be substituted without retraining.

##### 2) Agents for ECOs

ECOs represent a particularly promising application for LLM-based agents. ECOs, such as fixing setup and hold violations or resolving design rule violations, are typically addressed through manual, iterative interventions by expert designers. These tasks require understanding both local problem regions (e.g., a timing-critical path or a DRC hotspot) and global design context, including constraints and downstream impacts.

LLMs are well-suited to this setting because they can iteratively reason over rich contextual information and learn from prior attempts. By interacting directly with EDA tools, an agent can observe how specific edits affect timing or rule violations, refine its strategy, and converge toward a valid solution. Unlike traditional ECO scripts or greedy heuristics, an LLM-based agent can adapt its behavior based on design-specific characteristics and accumulated experience across iterations. Existing efforts to explore LLMs for DRC fixing illustrate the feasibility of this approach [52], suggesting that ECO automation may be one of the most impactful near-term applications of agentic LLMs in physical design.

##### E. Agents for EDA tool source code generation

LLMs have shown significant potential for algorithm discovery [46]. Recent advances in context engineering and modular adaptation, and code generation enable LLMs to generate EDA tool source code based on feedback from design metrics. This enables the idea of *design-tool co-evolution* in EDA, potentially yielding novel custom-tailored EDA tools for each.

An emerging approach is to build specialized EDA agents by dynamically providing structured context—such as tool documentation, design rules, constraints, and code abstractions—to existing coding agents. Within this framework, LLMs can directly modify EDA tool heuristics, generate code changes, and explore alternative algorithmic strategies tailored to a specific design instance as shown in Fig. 5.

The work in [53] describes this vision through OpenROAD-evolve, which proposes a design–tool co-exploration loop driven by LLM-based coding agents. In this paradigm, an agent analyzes both the design characteristics and the OpenROAD codebase, proposes targeted modifications to core heuristics, recompiles the tool, and evaluates the impact of these changes on downstream QoR. The design is then re-optimized using the updated tool, and the process iterates. This enables heuristics to be specialized to individual designs rather than remaining fixed across workloads.

AuDoPEDA [18] formalizes a closed-loop autonomous coding pipeline that systematically improves QoR for the open-source OpenROAD EDA stack by combining documentation, planning, and execution. It parses the OpenROAD codebase into an AST-based dependency graph and structured docs, then uses retrieval-augmented planning to propose improvement ideas and translate them into concrete code patches with tests. These patches are compiled and benchmarked in instrumented flows, with QoR feedback driving an iterative, guardrailed self-correction loop that validates and refines changes, thus converging on the idea of LLM-based code evolution in EDA.

Another recent work [17], inspired by AlphaEvolve, has also demonstrated the feasibility of extending LLM-driven code evolution to full repository scale. SATLUTION [17] is an example of such a framework, applying LLM-based evolutionary optimization to Boolean Satisfiability (SAT), a canonical NP-complete problem central to both theory and practice. SATLUTION orchestrates multiple LLM agents to evolve entire solver repositories comprising hundreds of files and tens of thousands of lines of C/C++ code, under strict correctness guarantees and distributed runtime feedback.

Together, efforts such as AuDoPEDA and SATLUTION highlight the potential of agentic LLMs to enable design–tool co-exploration, a direction that is directly relevant to future EDA tool development [19].

#### IV. CHALLENGES, IMPLICATIONS, AND OPPORTUNITIES

**Challenges.** While LLM-based approaches have demonstrated significant promise across a range of physical design tasks, their practical adoption poses challenges.

- **Non-deterministic behavior.** LLMs inherently exhibit non-deterministic outputs, where identical prompts can yield different responses across runs. This behavior complicates reproducibility, debugging, and verification in physical design workflows that demand strict correctness, traceability, and auditability.
- **Expensive agentic optimization loops.** Agent-based approaches often rely on repeated invocation of full EDA

tool flows, making optimization loops computationally expensive and time-consuming. This limits scalability and motivates the use of surrogate models, early-exit strategies, proxy metrics, and hierarchical evaluation to reduce runtime overhead.

- **Context window limitations.** Physical design artifacts such as DEF, LEF, GDS, timing reports, and detailed tool logs frequently span gigabytes, far exceeding the context capacity of current LLMs. Effectively summarizing, structuring, and selectively exposing relevant information—without losing critical design semantics—remains an open research challenge.
- **Rapid evolution and compatibility.** Both LLMs and open-source EDA tools are evolving rapidly. New LLM architectures and capabilities appear frequently, raising concerns around backward compatibility and the stability of agentic workflows. At the same time, tools such as OpenROAD undergo continuous development, with evolving APIs, data formats, and internal heuristics. LLM-based agents must therefore be designed with modular abstractions and robust interfaces to remain compatible with advancing models and changing toolchains.

**Implications.** These challenges have important implications for the design and evaluation of LLM-based physical design systems. Reproducibility and correctness must be treated as first-class concerns, requiring deterministic execution paths, and explicit tool-based validation. Evaluation methodologies must also account for runtime and resource cost, as QoR improvements achieved through expensive agentic loops may not translate to practical deployment. Finally, the rapid co-evolution of LLMs and EDA tools implies that static benchmarks and one-off demonstrations are insufficient; instead, continuous integration, regression testing, and version-aware evaluation are needed to ensure sustained progress.

**Opportunities.** At the same time, these challenges open significant research opportunities. Improved context management—through structured representations—can enable LLMs to reason effectively over large-scale physical design artifacts. Moreover, the rapid evolution of both LLMs and open-source EDA tools creates an opportunity to design adaptive, modular agentic frameworks that co-evolve with their underlying models, toolchains, designs, and technology.

#### V. CONCLUSION

This paper presented the evolution of LLM usage in physical design, tracing how LLMs have progressed from basic interfaces to more capable assistants and agent-based workflows. Grounded in representative open-source EDA efforts, this evolution highlights both the growing potential of LLMs to interact with complex physical design tools and the practical constraints that shape their use in real design flows. The observed trajectory of LLM usage suggests a natural extension beyond assistance, enabling more ambitious applications, such as ECO automation, agentic optimization, and algorithm discovery within physical design tools.

## REFERENCES

- [1] T. Ajayi, *et al.*, “INVITED: Toward an Open-Source Digital Flow: First Learnings from the OpenROAD Project,” in *Proc. DAC*, 2019.
- [2] T. Ajayi, *et al.*, “OpenROAD: Toward a Self-Driving, Open-Source Digital Layout Implementation Tool Chain,” in *Proc. GOMACTech*, 2019.
- [3] “OpenROAD.” <https://github.com/The-OpenROAD-Project/OpenROAD>, 2026.
- [4] “OpenROAD-flow-scripts.” <https://github.com/The-OpenROAD-Project/OpenROAD-flow-scripts>, 2026.
- [5] Y. Lu, *et al.*, “A Comprehensive Survey of Datasets for Large Language Model Evaluation,” in *Proc. ICTC*, 2024.
- [6] OpenAI, “ChatGPT.” <https://openai.com/chatgpt>, 2024.
- [7] A. Radford, *et al.*, “Improving Language Understanding by Generative Pre-Training,” in *OpenAI Blog*, 2018.
- [8] Y. Li, *et al.*, “Competition-Level Code Generation with AlphaCode,” in *Science*, vol. 378, 2022.
- [9] GitHub, “GitHub Copilot.” <https://github.com/features/copilot>, 2021.
- [10] M. Chen, *et al.*, “Evaluating Large Language Models Trained on Code,” *arXiv preprint*, 2021.
- [11] Anthropic, “Claude Code.” <https://github.com/anthropics/claude-code>, 2023.
- [12] B.-Y. Wu, *et al.*, “EDA Corpus: A Large Language Model Dataset for Enhanced Interaction with OpenROAD,” in *Proc. LAD*, 2024.
- [13] U. Sharma, *et al.*, “OpenROAD-Assistant: An Open-Source Large Language Model for Physical Design Tasks,” in *Proc. MLCAD*, 2024.
- [14] H. Wu, *et al.*, “ChatEDA: A Large Language Model Powered Autonomous Agent for EDA,” *IEEE T. Comput. Aid. D.*, vol. 43, no. 10, 2024.
- [15] A. Ghose, *et al.*, “ORFS-agent: Tool-Using Agents for Chip Design Optimization,” in *Proc. MLCAD*, 2025.
- [16] B.-Y. Wu, *et al.*, “OpenROAD Agent: An Intelligent Self-Correcting Script Generator for OpenROAD,” in *Proc. ICLAD*, 2025.
- [17] C. Yu, *et al.*, “Autonomous Code Evolution Meets NP-Completeness,” *arXiv preprint*, 2025.
- [18] A. Ghose, *et al.*, “Automated QoR improvement in OpenROAD with coding agents,” *arXiv preprint*, 2026.
- [19] A. Ghose, *et al.*, “Invited: Agentic AI for Physical Design R&D: Status and Prospects,” in *Proc. ISPD*, 2026.
- [20] A. B. Kahng, *et al.*, “SI for Free: Machine Learning of Interconnect Coupling Delay and Transition Effects,” in *Proc. SLIP*, 2015.
- [21] H. Yang, *et al.*, “Imbalance Aware Lithography Hotspot Detection: A Deep Learning Approach,” *J. Micro/Nanolith. MEMS MOEMS*, vol. 16, no. 3, 2017.
- [22] W.-T. J. Chan, *et al.*, “Routability optimization for industrial designs at sub-14nm process nodes using machine learning,” in *Proc. ISPD*, 2017.
- [23] Z. Xie, *et al.*, “RouteNet: Routability prediction for Mixed-Size Designs Using Convolutional Neural Network,” in *Proc. ICCAD*, 2018.
- [24] V. A. Chhabria, *et al.*, “Template-based PDN Synthesis in Floorplan and Placement Using Classifier and CNN Techniques,” in *Proc. ASP-DAC*, 2020.
- [25] V. A. Chhabria, *et al.*, “Thermal and IR Drop Analysis Using Convolutional Encoder-Decoder Networks,” in *Proc. ASP-DAC*, 2021.
- [26] J.-G. Lin, *et al.*, “DRC Violation Prediction with Pre-global-routing Features Through Convolutional Neural Network,” in *Proc. GLSVLSI*, 2023.
- [27] Y.-C. Lu, *et al.*, “RL-Sizer: VLSI Gate Sizing for Timing Optimization Using Deep Reinforcement Learning,” in *Proc. DAC*, 2022.
- [28] W. Jiang, *et al.*, “IR-Aware ECO Timing Optimization Using Reinforcement Learning,” in *Proc. MLCAD*, 2024.
- [29] C.-K. Cheng, *et al.*, “Assessment of Reinforcement Learning for Macro Placement,” in *Proc. ISPD*, 2023.
- [30] “Synopsys DSO.ai.” <https://www.synopsys.com/ai/ai-powered-eda/dso-ai.html>, 2026.
- [31] “Cadence Cerebrus.” [https://www.cadence.com/en\\_US/home/tools/digital-design-and-signoff/soc-implementation-and-floorplanning/cerebrus-intelligent-chip-explorer.html](https://www.cadence.com/en_US/home/tools/digital-design-and-signoff/soc-implementation-and-floorplanning/cerebrus-intelligent-chip-explorer.html), 2026.
- [32] S. Liu, *et al.*, “RTLCoder: Outperforming GPT-3.5 in Design RTL Generation with Our Open-Source Dataset and Lightweight Solution,” in *Proc. LAD*, 2024.
- [33] K. Chang, *et al.*, “A Data-Centric Chip Design Agent Framework for Verilog Code Generation,” *ACM T. Des. Automat. El.*, vol. 30, no. 6, 2025.
- [34] S. Thakur, *et al.*, “Verigen: A large language model for verilog code generation,” *ACM T. Des. Automat. El.*, vol. 29, Apr. 2024.
- [35] M. Liu, *et al.*, “ChipNeMo: Domain-Adapted LLMs for Chip Design,” *arXiv preprint*, 2023.
- [36] “Cadence JedAI.” [https://www.cadence.com/en\\_US/home/solutions/cadence-jedai-solution.html#](https://www.cadence.com/en_US/home/solutions/cadence-jedai-solution.html#), 2026.
- [37] A. B. Kahng, “Looking Into the Mirror of Open Source: Invited Paper,” in *Proc. ICCAD*, 2019.
- [38] A. B. Kahng, “A Mixed Open-Source and Proprietary EDA Commons for Education and Prototyping,” in *Proc. ICCAD*, 2022.
- [39] L. Ouyang, *et al.*, “Training Language Models to Follow Instructions with Human Feedback,” in *Proc. NeurIPS*, 2022.
- [40] E. J. Hu, *et al.*, “LoRA: Low-Rank Adaptation of Large Language Models,” in *Proc. ICLR*, 2022.
- [41] S. Yao, *et al.*, “ReAct: Synergizing Reasoning and Acting in Language Models,” in *Proc. ICLR*, 2022.
- [42] S. Yao, *et al.*, “Tree of Thoughts: Deliberate Problem Solving with Large Language Models,” in *Proc. NeurIPS*, 2023.
- [43] E. Zelikman, *et al.*, “Quiet-STaR: Language Models Can Teach Themselves to Think Before Speaking,” *arXiv preprint*, 2024.
- [44] B. Romera-Paredes, *et al.*, “Mathematical Discoveries from Program Search with Large Language Models,” *Nature*, vol. 625, no. 7995, 2024.
- [45] J. Lehman, *et al.*, “Evolution through Large Models,” *arXiv preprint*, 2022.
- [46] A. Novikov, *et al.*, “AlphaEvolve: A Coding Agent for Scientific and Algorithmic Discovery,” *arXiv preprint*, 2025.
- [47] Y. Pu, *et al.*, “Customized Retrieval Augmented Generation and Benchmarking for EDA Tool Documentation QA,” *IEEE T. Comput. Aid. D.*, vol. 44, no. 12, 2025.
- [48] A. Kaintura, *et al.*, “ORAssistant: A Custom RAG-based Conversational Assistant for OpenROAD,” *arXiv preprint*, 2024.
- [49] G. Pasandi, *et al.*, “JARVIS: A Multi-Agent Code Assistant for High-Quality EDA Script Generation,” *arXiv preprint*, 2025.
- [50] T. Zhang, *et al.*, “RAFT: Adapting Language Model to Domain Specific RAG,” *arXiv preprint*, 2024.
- [51] X. Li, *et al.*, “iEDA: An Open-source infrastructure of EDA,” in *Proc. ASP-DAC*, 2024.
- [52] C.-C. Chang, *et al.*, “DRC-Coder: Automated DRC Checker Code Generation Using LLM Autonomous Agent,” in *Proc. ISPD*, 2025.
- [53] V. A. Chhabria, *et al.*, “IEEE DATC RDF-2025: Enabling an EDA Research Ecosystem,” in *Proc. ICCAD*, 2025.