

The Dawn of Agentic EDA: A Survey of Autonomous Digital Chip Design

Zelin Zang¹, Yuhang Song², Bingo Wing-Kuen Ling¹, Aili Wang², and Fuji Yang*¹

¹Center for the Integrated Circuits and Artificial Intelligence, Tsientang Institute for Advanced Study

²ZJUI Institute, Zhejiang University

December 30, 2025

Abstract

This survey provides a comprehensive overview of the integration of Generative AI and Agentic AI within the field of Digital Electronic Design Automation (EDA). The paper first reviews the paradigmatic evolution from traditional Computer-Aided Design (CAD) to AI-assisted EDA (AI4EDA), and finally to the emerging AI-Native and Agentic design paradigms. We detail the application of these paradigms across the digital chip design flow, including the construction of agentic cognitive architectures based on multimodal foundation models, frontend RTL code generation and intelligent verification, and backend physical design featuring algorithmic innovations and tool orchestration. We validate these methodologies through integrated case studies, demonstrating practical viability from microarchitecture definition to GDSII. Special emphasis is placed on the potential for cross-stage feedback loops where agents utilize backend PPA metrics to autonomously refine frontend logic. Furthermore, this survey delves into the dual-faceted impact on security, covering novel adversarial risks, automated vulnerability repair, and privacy-preserving infrastructure. Finally, the paper critically summarizes current challenges related to hallucinations, data scarcity, and black-box tools, and outlines future trends towards L4 autonomous chip design. Ultimately, this work aims to define the emerging field of Agentic EDA and provide a strategic roadmap for the transition from AI-assisted tools to fully autonomous design engineers.

*Corresponding Author

Contents

1	Introduction: Paradigm Reconstruction	3
1.1	The Productivity Gap and Automation Bottlenecks	3
2	Foundations: The Brain and Representation	4
2.1	Circuit Foundation Models (CFMs)	5
2.2	Domain-Adapted Large Language Models	5
2.3	Industrial State-of-the-Art: A Reference Framework	6
2.4	Agentic Cognitive Architectures	6
3	Frontend: RTL & Verification (From Generation to Autonomy)	7
3.1	Iterative Code Generation and Repair: From Syntax to Semantics	8
3.1.1	The Syntactic Loop: Compiler-Guided Feedback	8
3.1.2	The Semantic Loop: AST-Based Waveform Tracing	8
3.2	Hallucination Mitigation and Formal Alignment	9
3.3	Agentic Verification and Debugging	10
3.4	Case Studies: Microarchitecture & RTL	10
4	Backend: Physical Design (Algorithms & Tool Orchestration)	11
4.1	The Layout Paradigm Shift: From Sequential RL to Generative Diffusion	11
4.2	Graph Learning for Transferable Physics	12
4.3	Closing the Loop: Cross-Stage Optimization	13
4.4	Case Study: Autonomous Flow Orchestration	14
5	Security, Trust & Infrastructure	14
5.1	The Adversarial Dynamic: Agent vs. Agent Security	14
5.2	Privacy-Preserving Collaboration: Federated Learning	15
5.3	Next-Generation Benchmarks and Datasets	15
6	Conclusion and Future Outlook	15
6.1	Summary of Challenges	15
6.2	Future Trends: Towards AI-Native Autonomy	17

1 Introduction: Paradigm Reconstruction

The integrated circuit (IC) design industry is currently navigating a historic inflection point. For decades, Electronic Design Automation (EDA) has evolved linearly from manual layout to Computer-Aided Design (CAD), and subsequently to algorithm-based automation targeting specific logic synthesis and physical design tasks. While machine learning has recently been integrated to enhance specific point tools, the explosive emergence of Large Language Models (LLMs) and Agentic AI marks a profound transition from “automation assistance” to “autonomous design” [1]. This evolutionary trajectory, as illustrated in Fig. 1, promises to move beyond static optimization algorithms toward cognitive systems capable of reasoning, planning, and tool orchestration.

1.1 The Productivity Gap and Automation Bottlenecks

The Complexity Explosion under Moore’s Law. Although the marginal benefits of Moore’s Law face physical scaling challenges, modern System-on-Chip (SoC) designs have escalated to the scale of hundreds of billions of transistors. Human design productivity, however, has failed to keep pace with this complexity, creating a critical “Productivity Gap.” Empirical data suggests that verification tasks alone often consume 60% to 70% of the entire development cycle, while design costs surge dramatically with each advancing process node [2]. Traditional script-based automation is becoming increasingly insufficient to bridge this gap, necessitating the paradigm shift shown in the later stages of our evolutionary timeline.

From AI4EDA to AI-Native EDA. In recent years, both academia and industry have extensively explored “AI for EDA” (AI4EDA). However, Chen et al. [3] argue that most current AI4EDA approaches merely adapt models from Computer Vision (CV) or Natural Language Processing (NLP) to circuit tasks. As depicted in Stage 2 of Fig. 1 (often corresponding to L2 Copilot systems), these approaches often act as “patches” on existing software interfaces rather than foundational reconstructions. This aligns with the broader vision of “Third-Generation AI” proposed by Zhang et al. [4], which emphasizes the integration of knowledge, data, and reasoning—a prerequisite for the autonomous agents discussed in this survey.

Terminology and Autonomy Levels. To keep terminology consistent across sections, we refer to autonomy levels as **L0–L5** following the taxonomy summarized in Table 6. In brief, **L2** corresponds to “Copilot”-style assistance, while **L3+** denotes agentic systems with multi-step execution loops that can operate with reduced human intervention.

In contrast, the concept of “AI-Native EDA” (Stage 3 in Fig. 1, enabling L3 agentic workflows) has emerged as a necessary evolution. It advocates for positioning AI at the core of the design process, relying on multimodal Circuit Foundation Models (CFMs) capable of simultaneously comprehending netlists, Register-Transfer Level (RTL) code, and physical layouts [3, 5].

Relationship to Prior Work. This survey builds directly upon the vision of “AI-Native EDA” established by Chen et al. [3]. While their work laid the *infrastructural foundation*—proposing Circuit Foundation Models (CFMs) as the unified representation—this paper focuses on the *cognitive execution layer*. We explore how Agentic workflows rise above this foundation to transition from static perception to dynamic, autonomous action, effectively evolving from the “Brain” (Model) to the “Engineer” (Agent).

Intelligent Design 4.0: From Tools to Agents. A sharper distinction must be drawn between traditional automation and the emerging autonomy. Previous EDA tools, even those enhanced by ML (Stage 2), function as **Automation**: they are deterministic point solutions requiring essentially a “human-in-the-loop” to bridge disconnected tasks. In contrast, Intelligent Design 4.0 represents **Autonomy**: agents that perceive global flow contexts, plan multi-step strategies, and self-correct, shifting the human role to a supervisory “on-the-loop” position.

To put it sharply: traditional AI optimized the “wrench” (the tool), whereas Agentic AI aims to automate the “engineer” using the wrench. As shown in the final stage of Fig. 1, these autonomous agents—equipped with the “Reasoning-Acting-Reflecting” loop—can inter-

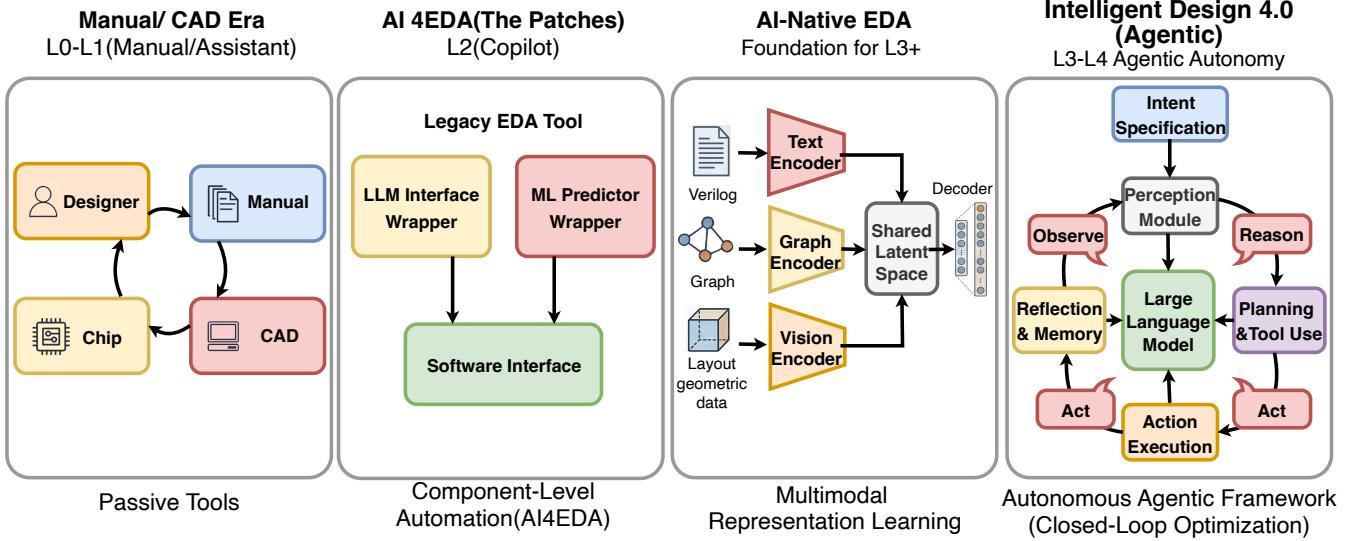


Figure 1: Evolution of IC Design Paradigms toward Autonomy. The figure depicts a four-stage trajectory: evolving from passive (1) Manual/CAD tools and (2) fragmented AI4EDA patches, to a unified (3) AI-Native EDA foundation, and finally realizing (4) Intelligent Design 4.0 driven by autonomous agents with reasoning-acting-reflecting loops.

Table 1: Evolution of IC Design Paradigms: From Automation to Autonomy.

Paradigm	Core Technology	Human Role	Key Characteristic	Representative Concept
Manual/CAD	Geometric Algorithms	Operator	Deterministic Execution	Script-based Automation
extbf{AI4EDA}	CV/NLP Models (Patches)	Controller	Point Tool Optimization	Adaptation of Non-EDA Models
AI-Native EDA	Multimodal Foundation Models	Supervisor	Cross-Stage Transfer Knowledge	Circuit Foundation Models (CFMs)
Intelligent Design 4.0	Autonomous Agents	Intent Specifier	Reasoning & Self-Reflection	Agent-in-the-loop

interpret ambiguous intent, orchestrate complex tool chains, and learn from execution logs to iteratively improve designs.

Revolution or Hype? Despite this ambitious vision, the application of large models in hardware design faces severe challenges, including hallucinations, data scarcity, and the opacity of black-box EDA tools. He et al. [6] provide a critical assessment of LLMs across code generation, verification, and optimization, noting that while progress is exciting, a significant gap remains before achieving the true industrial-grade “autopilot” depicted in our vision. Furthermore, Xu et al. [2] emphasize that the community must move beyond superficial metrics and establish rigorous benchmarks focused on end-to-end Power, Performance, and Area (PPA) to determine whether this technological shift represents a genuine revolution.

Scope of this Survey. Given the fundamental differences in design paradigms between digital and analog circuits—where the former relies heavily on logic synthesis and discrete optimization, while the latter depends on continuous physical equations—this survey explicitly focuses on Digital Chip Design. We explore how agentic workflows are transforming the standard RTL-to-GDSII flow, leaving the distinct challenges of analog automation to future dedicated reviews.

2 Foundations: The Brain and Representation

Building an autonomous chip design system requires two fundamental capabilities: the “Perception” to accurately represent the multimodal structure of digital circuits, and the “Brain” to reason, plan,

and execute complex design flows. This section explores the architectural foundations of Agentic EDA, transitioning from static representation learning to dynamic cognitive architectures.

2.1 Circuit Foundation Models (CFMs)

While Large Language Models (LLMs) excel at processing textual hardware description languages (HDLs), a circuit is intrinsically a multimodal entity comprising netlist graphs, logical truth tables, and geometric layouts. To bridge the gap between semantic understanding and physical reality, Circuit Foundation Models (CFMs) have emerged as the perceptual layer of AI-Native EDA [3].

Multimodal Representation Learning. Fang et al. [5] categorize CFMs into encoder-based and decoder-based architectures, emphasizing that effective models must unify text, graph, and layout modalities. *CircuitFusion* [7] represents a pioneering effort in this domain. By employing a **Contrastive Learning** strategy that aligns the latent embeddings of hardware code, structural graphs, and functional summaries, it captures the intrinsic properties of circuits—such as parallel execution and functional equivalence—enabling zero-shot transfer across diverse downstream tasks. General unsupervised representation frameworks [8] further support the theoretical alignment of these heterogeneous modalities. This builds upon earlier concepts of Large Circuit Models [9], which first identified the need for unified representations to handle the growing complexity of modern SoCs.

Scalable Graph Learning. While multimodal representation ensures semantic alignment across domains, practical deployment requires handling massive graphs without computational collapse. A critical challenge in circuit representation is scalability; modern SoCs contain billions of nodes, causing conventional Graph Neural Networks (GNNs) to suffer from memory bottlenecks and over-smoothing [10]. To address this, *DeepGate4* [11] introduces a scalable graph transformer tailored for logic synthesis. It utilizes an update strategy that reduces memory complexity to sub-linear levels, significantly outperforming state-of-the-art methods on large-scale benchmarks. Similarly, Luo et al. [12] propose *DE-HNN*, a directed equivariant hypergraph neural network, to better model the high-order interactions in netlists that standard graphs fail to capture. Furthermore, recent advancements in manifold graph embedding [13, 14, 15] and heterophilic graph learning [16] offer promising directions for capturing the complex, non-Euclidean geometry of circuit netlists, leveraging geometric structure preservation [17]. Finally, to aid human understanding of these latent spaces, techniques for structure-preserving visualization [18] and explainable dimension reduction [19] are becoming increasingly important.

2.2 Domain-Adapted Large Language Models

General-purpose LLMs struggle with the verbose syntax and proprietary protocols of hardware design. *Domain-Adaptive Pre-training (DAPT)* and tokenizer optimization are essential to adapt the “Brain” for EDA.

Tokenizer Optimization for Verilog. Standard tokenizers (e.g., BPE) fragment Verilog code inefficiently; in practice, general LLM tokenizers often split common keywords like `always` or `posedge` into multiple sub-tokens. To address this, domain-specific models employ Custom Tokenizers that:

1. **Merge Frequent Keywords:** Treat high-frequency Verilog constructs (e.g., `module`, `assign`, `always_ff`) as single tokens.
2. **Preserve Indentation:** Encode whitespace structures critical for readability as dedicated tokens.

In practice, such optimizations can substantially reduce token counts for RTL, improving effective context utilization when analyzing long designs.

Dynamic Knowledge Integration: The Necessity of RAG. While Domain-Adaptive Pre-training (DAPT) injects general knowledge into the model’s weights (parametric knowledge), it fundamentally fails to address the dynamic and proprietary nature of industrial EDA. Process Design

Kits (PDKs), internal design methodologies, and rapid tool updates occur on a weekly basis, making continuous pre-training impractical. Consequently, **Retrieval-Augmented Generation (RAG)** has transitioned from an auxiliary enhancement to a mandatory component for industrial-grade EDA agents.

RAG bridges the gap by allowing the LLM to query external, non-parametric knowledge bases during inference. This is critical for grounding agent actions in ground-truth specifications, such as retrieving a voltage threshold from a foundry technology file before formulating a power optimization strategy.

Technical Challenges in EDA-Specific RAG. Implementing RAG for hardware design presents unique challenges beyond generic text retrieval, primarily due to the **heterogeneous and structured** nature of EDA documentation. First, critical information often resides not in prose but in dense tables, such as non-linear delay models (NLDM) in Liberty (.lib) files or intricate register maps in datasheets. Standard text embedding models often fail to capture the row-column relationships crucial for interpreting these values correctly. Second, engineering PDFs contain multi-column layouts, embedded diagrams, and cross-page tables that break standard parsing tools. Naive chunking strategies (e.g., fixed token windows) often sever a table’s header from its data rows or disconnect a design rule from its essential footnotes, rendering the retrieved context useless for the agent.

2.3 Industrial State-of-the-Art: A Reference Framework

To ground academic research in industrial reality, it is essential to benchmark against leading commercial AI-driven tools.

- **Synopsys DSO.ai:** Utilizes Reinforcement Learning (RL) to explore the massive state space of physical design parameters. Its state space S includes congestion maps and timing slack distributions, while its reward function R is a weighted sum of PPA metrics ($R = w_1 \cdot TNS + w_2 \cdot Power + w_3 \cdot Area$).
- **Cadence Cerebrus:** Distinguishes itself with *Transfer Learning*, enabling the system to apply optimization strategies learned from previous 7nm designs to new 5nm projects, significantly accelerating convergence.
- **Google AlphaChip:** Revolutionized macro placement using Edge-Based Graph Neural Networks (GNNs) to learn the embeddings of circuit connectivity, demonstrating that learning-based approaches can outperform human experts in floorplanning.

2.4 Agentic Cognitive Architectures

The transition from a passive “Copilot” to an active “Autopilot” relies on the cognitive architecture of the agent. This involves equipping LLMs with the ability to reason about design states, plan multi-step actions, and collaborate within a society of agents.

Reasoning and Planning (ReAct & CoT). Simple input-output generation is insufficient for complex flows. Agents now employ the *Reasoning + Acting* (ReAct) paradigm [20]. For instance, when encountering a synthesis error, an agent does not merely guess a fix; it first generates a thought trace (“The log indicates a combinational loop”), executes a tool (“Run timing analysis to locate the path”), observes the output, and then plans the correction. This Chain-of-Thought (CoT) process transforms the LLM from a generator into a reasoner [21].

Multi-Agent Collaboration Patterns. To overcome the context limitations and bias of single agents, multi-agent systems (MAS) are becoming the standard [1]. One prevalent pattern is **Hierarchical Planning**, as seen in *ChatEDA* [22], where a controller agent decomposes high-level specifications into sub-tasks and dispatches them to specialized tool-execution agents. Another key approach is the **Generator-Critic Loop**. In frameworks like *AnaFlow* [23], different agents assume distinct roles—one generates the circuit topology while another critiques it against design constraints,

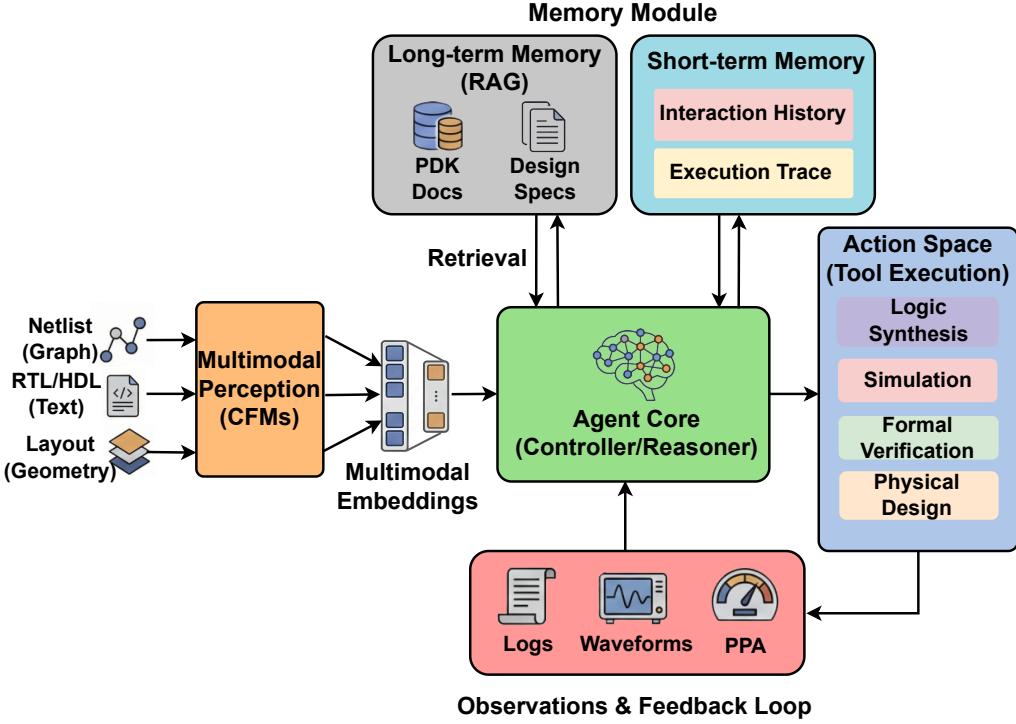


Figure 2: The cognitive architecture of an EDA Agent. It integrates Multimodal Perception (via CFMs), Retrieval-Augmented Memory (RAG), and a ReAct-based Tool Execution loop to interact with standard EDA toolchains.

ensuring robust and explainable decision-making. This aligns with emerging logic-driven multi-agent frameworks [24] that integrate rigorous reasoning paths to handle complex domain-specific constraints. Finally, for subjective trade-offs such as PPA balancing, agents employ **Debate and Consensus** mechanisms, where multiple agents propose competing solutions and debate their merits to escape local optima [25].

With these foundations—multimodal representations, domain-adapted models, and agentic execution architectures—we next examine how autonomy concretely manifests in the *frontend* of the digital flow, from RTL generation to verification loops.

3 Frontend: RTL & Verification (From Generation to Autonomy)

Frontend design—the translation of natural language specifications into functional Register-Transfer Level (RTL) code and its subsequent verification—represents the most mature domain for Agentic AI in EDA. The research trajectory in this field has evolved distinctly from naive “one-shot generation” [34] to robust “iterative repair” and, more recently, to “autonomous verification” agents. According to the latest benchmarks from *Revisiting VerilogEval* [35], state-of-the-art models like GPT-4o have achieved a 63% pass rate on specification-to-RTL tasks, yet a significant gap remains in handling complex, multi-module designs, necessitating the agentic workflows described below.

The Paradigm Shift: From Copilot to Agent. It is crucial to distinguish between “Copilot” and “Agentic” paradigms. Copilot systems (L2) function as intelligent autocomplete engines, relying on human intent for every step. In contrast, Agentic systems (L3+) operate as autonomous orchestrators. They possess a “Reasoning-Acting-Reflecting” loop, allowing them to self-correct errors and invoke tools without explicit human intervention for every sub-task.

In the remainder of this section, we use the shorthand **L2** (Copilot) and **L3+** (agentic autonomy)

Table 2: Summary of Agentic Frameworks for Frontend Design (RTL Generation, Repair, and Verification).

Framework	Target Domain	Key Methodology / Innovation
<i>AutoChip</i> [26]	RTL Repair (Syntactic)	Feedback loop using compiler error logs (text) for syntax correction
<i>VeriAssist</i> [27]	RTL Repair (Functional)	Self-verification loop where the LLM generates its own testbench to validate logic
<i>VerilogCoder</i> [28]	Deep Semantic Repair	AST-based waveform tracing to translate signal mismatches into natural language
<i>CraftRTL</i> [29]	Data Augmentation	Synthetic data generation with injected errors for robust repair training
<i>HaVen</i> [30]	Hallucination Mitigation	Structured Instruction Chain-of-Thought (SI-CoT) for protocol alignment
<i>ASPEN</i> [31]	Datapath Optimization	Neuro-symbolic approach using E-Graphs for mathematically equivalent rewriting
<i>Saarthi</i> [32]	Verification Planning	Autonomous generation of verification plans and formal assertions
<i>DRC-Coder</i> [33]	Physical Verification	Multimodal agents (Vision + Text) for DRC deck debugging

consistent with Table 6.

3.1 Iterative Code Generation and Repair: From Syntax to Semantics

Industry adoption has revealed that one-shot generation is insufficient for the zero-tolerance nature of hardware design. Consequently, the “Generate-Compile-Feedback-Repair” loop has become the standard paradigm.

3.1.1 The Syntactic Loop: Compiler-Guided Feedback

AutoChip [26] established the baseline for this paradigm by constructing a text-based feedback loop. **Mechanism & Data Flow:** Instead of blindly regenerating code, AutoChip acts as a log parser. The data flow follows: *Code* \rightarrow *Compiler* \rightarrow *ErrorLog* \rightarrow *LLM*. It extracts specific error messages (e.g., `syntax error`, `port width mismatch`) from EDA tools like `iverilog` or `Yosys` and appends them to the original prompt, instructing the LLM to “fix only the identified lines.” While effective for ensuring syntactically valid code, this approach operates blindly regarding the circuit’s functionality. Similarly, *ChipGPT* [36] employs an “Output Manager” to sanitize the code before final output, further illustrating the trend of using auxiliary agents to enforce syntactic correctness.

3.1.2 The Semantic Loop: AST-Based Waveform Tracing

To address functional correctness, frameworks like *VeriAssist* [27] and *VerilogCoder* [28] introduce semantic repair loops that bridge the modality gap between textual RTL and binary simulation waveforms.

Mechanism & Data Flow: Unlike AutoChip’s text-only feedback, *VerilogCoder* operates on the simulation trace. When a testbench fails, the agent utilizes an Abstract Syntax Tree (AST) analysis to map the failing output signal back to its driving logic blocks. It traces the signal dependency chain (e.g., $output \leftarrow reg_A \leftarrow wire_B$) to locate the root cause of the value mismatch. Crucially, as discussed in [37], interpreting hardware states requires translation; VerilogCoder converts specific timestamp data into a natural language description (e.g., “Signal `ack` remained low at cycle 5 but was expected high”). This allows the agent to perform “Deep Semantic Repair” based on functional causality rather than just compiler complaints. *VeriAssist* complements this by introducing a “self-verification” mechanism, where the agent generates the testbench alongside the design, creating a self-contained simulation loop to validate its own logic before human review.

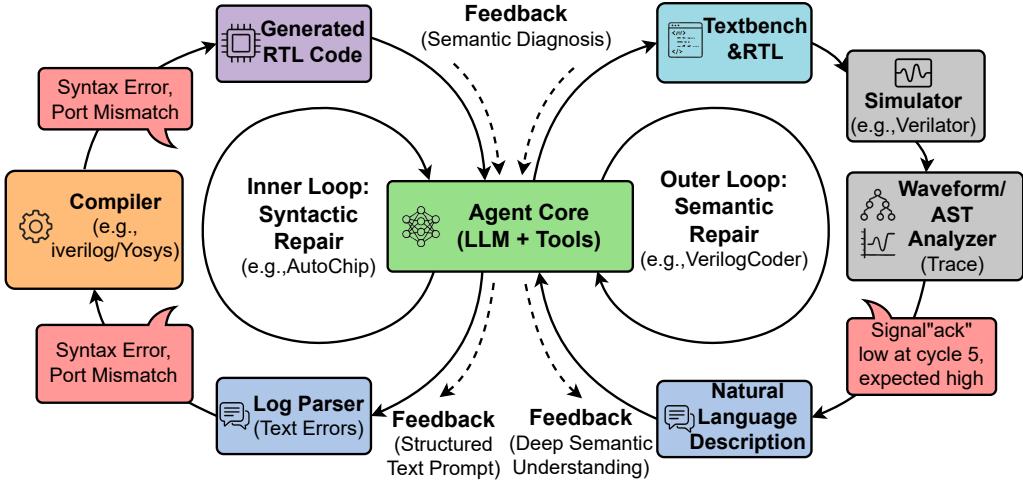


Figure 3: **The Dual-Loop Architecture of Agentic RTL Repair.** The *Inner Loop* (e.g., AutoChip [26]) relies on textual compiler logs to fix syntax. The *Outer Loop* (e.g., VerilogCoder [28]) bridges the modality gap by tracing simulation waveforms via AST analysis, converting signal mismatches into natural language feedback for semantic logic repair.

Data Infrastructure. The success of these repair agents relies on high-quality training data. *CraftRTL* [29] addresses data scarcity by using a “synthetic bug injection” strategy, creating massive paired datasets of “buggy code” and “fixed code.” Additionally, diffusion-based data augmentation techniques [38] are being adapted to generate high-fidelity synthetic training samples without requiring ground-truth pairs. Furthermore, *OpenLLM-RTL* [39] has recently released a large-scale open dataset including 80k instruction-code pairs and 7k verified samples, providing the necessary infrastructure to train these domain-specific repair models.

3.2 Hallucination Mitigation and Formal Alignment

While repair loops fix errors post-generation, preventing them requires aligning the LLM’s probabilistic nature with deterministic hardware protocols. *HaVen* [30] addresses this via a prompt engineering technique known as Structured Instruction Chain-of-Thought (SI-CoT).

Mechanism of SI-CoT. Standard LLMs often hallucinate timing violations because they generate code token-by-token without a global view of the clock cycle. SI-CoT mitigates this by enforcing a strict reasoning structure before any Verilog code is generated. The prompt explicitly requires the model to:

1. **Summarize the Protocol:** Extract key signal dependencies (e.g., “valid must wait for ready”).
2. **Describe State Transitions:** Output a natural language description of the FSM (e.g., “Transition from IDLE to TRANSMIT when start is high”).
3. **Generate Pseudo-Timing Diagrams:** Visualize signal interactions in text format.

This “Think-before-Code” constraint forces the model to ground its reasoning in engineering logic. For mathematically rigorous optimization, *ASPEN* [31] adopts a neuro-symbolic approach. It uses an LLM as a heuristic guide to explore the optimization space, while the actual rewriting is performed on an E-Graph (Equivalence Graph), guaranteeing that all optimization steps are mathematically equivalent and hallucination-free.

3.3 Agentic Verification and Debugging

Verification consumes over 60% of the design cycle. AI Agents are evolving from code assistants into autonomous “AI Verification Engineers.”

Neuro-Symbolic Verification. A critical bottleneck in current LLM-based verification is the high rate of semantic errors in generated assertions. Benchmarks like *AssertEval* [40] reveal that commercial LLMs produce semantically incorrect assertions 63% of the time. To address this, *SANGAM* [41] introduces a neuro-symbolic approach by modeling assertion generation as a Monte Carlo Tree Search (MCTS) problem. Instead of one-shot generation, it explores a tree of possible temporal operators, using simulation feedback to prune invalid branches. This shifts the paradigm from “Text Generation” to “Logic Search.”

Formal Feedback Integration. Moving beyond simulation, *VeriMaAS* [42] integrates Formal Verification (FV) directly into the agentic loop. Unlike simulation logs which only show *what* happened, FV tools provide a *Counter-Example Trace*—a precise sequence of states leading to a failure. VeriMaAS translates this trace into natural language, allowing the agent to “see” the exact causality of the bug, achieving a 7% higher pass rate than standard fine-tuning methods.

From RL to Agentic Planning. Traditionally, reinforcement learning (RL) was used to maximize coverage in verification, as seen in DDPG-based approaches [43]. However, RL struggles with understanding high-level specifications. *Saarathi* [32] demonstrates a shift to agentic planning, where the LLM parses English specifications to autonomously formulate verification plans and invoke formal tools to run assertions. *PRO-V-R1* [44] further enhances this by using an indirect strategy: the agent writes Python scripts to generate high-precision test vectors rather than generating the raw bits directly.

Multimodal Debugging. In physical verification, *DRC-Coder* [33] utilizes multimodal agents to generate and debug Design Rule Checking (DRC) decks. By combining visual layout analysis with textual rule reasoning, it achieves perfect F1 scores on standard cell benchmarks, significantly outperforming text-only prompting. For functional debugging, *FVDebug* [45] leverages Causal Graphs to drive agents in reverse-reasoning through complex dependency chains, accurately pinpointing the root cause of verification failures.

3.4 Case Studies: Microarchitecture & RTL

To illustrate the practical viability of Agentic EDA in the frontend, we examine two key case studies spanning microarchitecture definition and conversational RTL generation.

MCT-Explorer: AI in Microarchitecture Definition. Moving upstream from RTL, *MCT-Explorer* [46] demonstrates AI’s value in defining the microarchitecture itself. Determining parameters like cache size, issue width, and ROB depth is a massive non-convex optimization problem. By combining Monte Carlo Tree Search (MCTS) with Bayesian Optimization, MCT-Explorer navigates this high-dimensional space to find Pareto-optimal configurations. On the Gemmini SoC benchmark, it improved the Average Distance to Reference Set (ADRS) by 30.9% compared to traditional methods, proving that agents can optimize the “blueprint” before a single line of code is written.

ChipChat: The Feasibility of Conversational Design. *ChipChat* [47] represents the first successful demonstration of an AI-designed chip reaching tape-out. Through 100+ conversational turns, engineers guided an LLM to generate the Verilog for an 8-bit accumulator-based microprocessor. While it proved feasibility, the process relied heavily on human engineers to interpret error logs and prompt the model for fixes. This “Human-in-the-loop” limitation underscores the necessity for the autonomous feedback loops (e.g., AutoChip) discussed in Section 3.1.

Having covered frontend autonomy and its verification-centered feedback loops, we next turn to the backend physical design stage, where agents either learn placement/routing physics or orchestrate legacy toolchains to close PPA.

Table 3: Comparison of AI Approaches in Physical Design: Algorithm-Centric vs. Agent-Centric.

Category	Representative Method	Core Mechanism	Advantage/Limitation
RL	<i>AlphaChip</i> [48]	Sequential Decision Making	High training cost; weak zero-shot generalization
Generative	<i>Diffusion Placement</i> [49]	Denoising Process	Strong zero-shot generalization to unseen netlists
Frequency	<i>DCTdiff</i> [50]	Spectral Autoregression	Efficient inductive bias for spatial distribution
GNN	<i>TransPlace</i> [51]	Representation Learning	Transferable placement knowledge across blocks
Agentic	<i>ORFS-Agent</i> [52]	Tool Parameter Tuning	Operates legacy tools; mimics human PPA closure flow

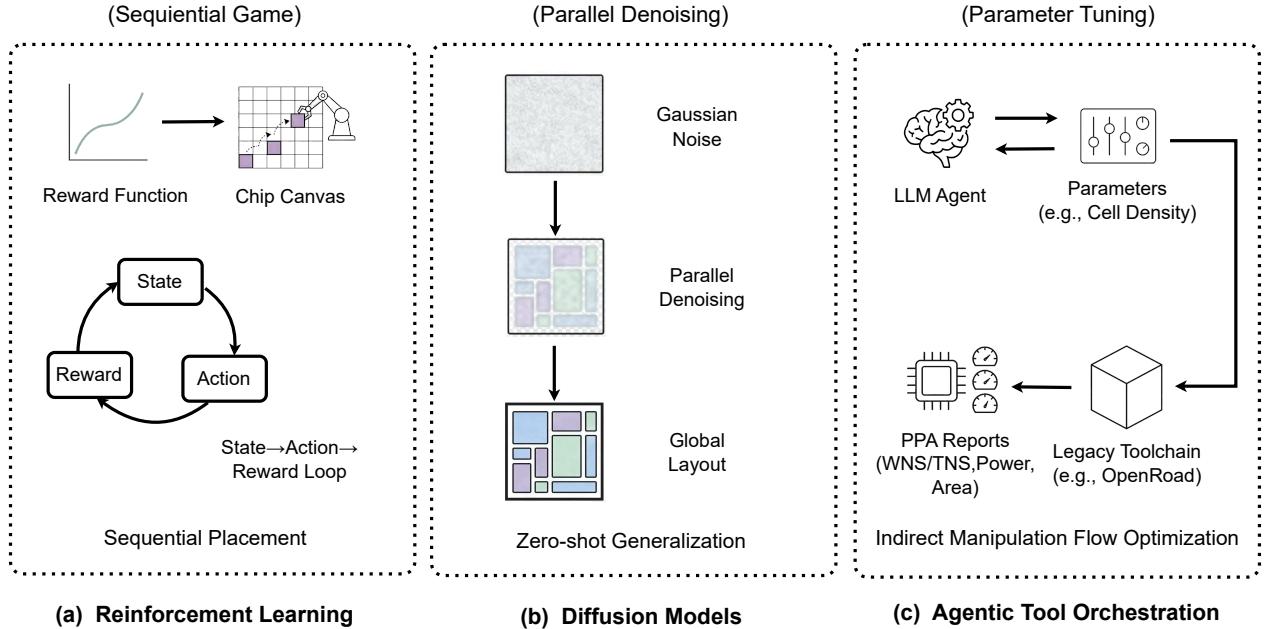


Figure 4: **Visual comparison of physical design paradigms:** (a) Reinforcement Learning treats placement as a sequential game; (b) Diffusion Models generate layouts via parallel denoising; (c) Agentic approaches optimize PPA by tuning toolchain parameters rather than direct geometric manipulation.

4 Backend: Physical Design (Algorithms & Tool Orchestration)

Physical design, particularly placement and routing (P&R), represents the most computationally intensive phase in the EDA flow. This domain is witnessing a fundamental dichotomy: *Algorithm-Centric* approaches are replacing heuristics with generative physics (Diffusion), while *Agent-Centric* approaches employ LLMs to orchestrate legacy tools.

4.1 The Layout Paradigm Shift: From Sequential RL to Generative Diffusion

For years, Reinforcement Learning (RL) dominated macro placement, epitomized by Google’s *AlphaChip* (formerly Circuit Training) [48]. However, the community is currently shifting towards Generative AI due to inherent scalability limits in RL.

The Bottleneck of Reinforcement Learning. RL methods formulate placement as a **Sequential Markov Decision Process (MDP)**. The agent places macros one by one, optimizing a reward function R (e.g., negative wirelength).

- **Mechanism Limitation:** Since the agent observes a partially placed state S_t to determine the next action A_t , the inference time grows linearly $O(N)$ with the number of macros.

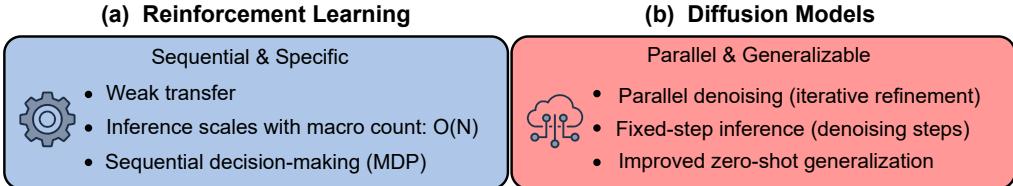


Figure 5: **Paradigm Shift in Macro Placement: From Sequential to Generative.** (a) **Reinforcement Learning** (e.g., AlphaChip) treats placement as a sequential game, where inference time grows linearly with macro count. (b) **Diffusion Models** (e.g., DCTdiff [50]) formulate placement as a parallel denoising process, learning the joint probability distribution of valid layouts to enable zero-shot generalization.

- **Generalization Issue:** As highlighted in [49], RL agents tend to overfit to the specific netlist topology seen during training. They memorize “solutions” rather than learning “placement rules,” necessitating expensive re-training (hundreds of GPU-hours) for every new chip design.

Diffusion Models: Learning the Distribution of Layouts. In contrast, Diffusion models [49, 50] redefine placement as a **Conditional Denoising** task. Instead of making sequential decisions, they learn the joint probability distribution $p_\theta(x|c)$ of valid cell coordinates x conditioned on the netlist c . **The Denoising Mechanism:**

1. **Training:** The model learns to reverse a forward diffusion process that gradually adds Gaussian noise to valid layouts until they become random noise.
2. **Inference (Placement):** Starting from pure noise, the model iteratively “denoises” the coordinates guided by the netlist connectivity gradients.

This paradigm allows for **Zero-Shot Generalization**: because the model learns the intrinsic spatial relationships of logic clusters (i.e., the underlying physics of connectivity), it can generate valid layouts for unseen netlists in sub-linear time, significantly outperforming RL in runtime efficiency.

Critical Analysis: The Precision Gap and Hybrid Workflows. Despite these advances, generative models face a “Precision Gap” in industrial adoption. While Diffusion is excellent for global structure (Macro Placement), it lacks the sub-pixel precision required for detailed standard cell placement and often produces outputs with Design Rule Check (DRC) violations. To bridge this gap, current pragmatic approaches employ a **Hybrid Workflow** [49, 53]. In this paradigm, the Diffusion model acts as a “Global Macro Planner,” utilizing its ability to escape local optima to determine the locations of large blocks. These coordinates are then fixed, and the task is handed over to differentiable analytical solvers (e.g., DREAMPlace).

Agents as Orchestrators, Not Just Replacements. It is a misconception that Agentic AI replaces algorithmic solvers or RL. Instead, it acts as a hierarchical *Orchestrator*. The Agent perceives the global design state and “calls” the diffusion model or analytical solver as a sub-routine, much like a human engineer uses a tool. This hierarchical approach combines the reasoning power of LLMs with the mathematical precision of specialized algorithms. For instance, frameworks like *TSCompiler* [54] demonstrate how efficient compilation frameworks can support dynamic-shape models, providing the necessary infrastructure for such adaptive agentic execution. Similarly, *ORFS-Agent* [52] exemplifies this by autonomously tuning tool parameters within the OpenROAD flow to close PPA, effectively mimicking a human backend engineer.

4.2 Graph Learning for Transferable Physics

While generative models handle coordinates, Graph Neural Networks (GNNs) are essential for encoding the topological “physics” of the netlist. A critical challenge is the **Heterogeneity** of circuits: netlists contain two distinct node types (cells and nets) with hyperedge connections.

Table 4: Empirical Comparison of Cross-Stage Feedback Frameworks.

Framework	Feedback Source	Optimization Goal	Key Empirical Result
<i>AutoChip</i> [26]	Compiler Logs	Syntax Correctness	89.6% Cost Reduction
<i>PEFA</i> [55]	VCD Waveforms	Functional Logic	Hierarchical Log Summarization
<i>VeriMaAS</i> [42]	Formal Verification	Property Safety	+7% Pass@1 vs. Fine-Tuning
<i>REvolution</i> [56]	PPA Reports	Power/Area	24.5% Power Reduction

TransPlace: Disentangling Topology and Geometry. *TransPlace* [51] addresses this by proposing a domain-specific GNN architecture.

- **Heterogeneous Message Passing:** Unlike standard GraphSAGE which assumes homogeneous nodes, TransPlace employs a bipartite update scheme. It alternates message passing between cell-nodes and net-nodes, effectively capturing the hypergraph structure of circuit connectivity.
- **Relative Position Encoding:** To solve the problem that standard GNNs are permutation invariant (ignoring spatial geometry), TransPlace introduces relative position encodings. This allows the model to learn “Cell-Flow”—predicting where a cell should move relative to its neighbors to minimize congestion.

This architecture enables **Transfer Learning**: a model trained on small RISC-V designs can successfully predict congestion hotspots on larger, unseen commercial TPUs, reducing the need for training from scratch.

4.3 Closing the Loop: Cross-Stage Optimization

The ultimate promise of Agentic EDA lies not just in optimizing individual stages, but in closing the feedback loop between frontend logic and backend physics. In traditional flows, a timing violation found during routing often requires a manual, heuristic-driven ECO (Engineering Change Order) or a complete RTL rewrite.

Evidence-Based Feedback Mechanisms. Recent empirical studies have validated the efficacy of this closed-loop approach. *AutoChip* [26] demonstrates that by integrating compiler feedback, the cost of code generation can be reduced by 89.6% while improving success rates by 5.8% compared to zero-shot baselines. This proves the economic viability of hybrid strategies that combine smaller models for generation with larger models for feedback-driven repair. Furthermore, *PEFA* [55] addresses the “information overload” problem of raw EDA logs by introducing a specialized “Log Summarizer Agent” that distills thousands of lines of simulation output into structured, actionable insights for the reasoning agent.

PPA Optimization via Evolutionary Agents. Beyond correctness, agents are now capable of optimizing Power, Performance, and Area (PPA). *REvolution* [56] introduces an evolutionary computation framework where agents maintain a population of designs, iteratively applying “Refactor” and “Fusion” operators. Empirical results show a 24.5% reduction in power consumption and a 24.0% increase in pass rates on the RTLLM benchmark, providing solid evidence that agentic workflows can surpass human-written baselines in multi-objective optimization.

Agentic Feedback Loops. Emerging frameworks are enabling agents to bridge this gap autonomously. By parsing backend reports (e.g., timing slack, congestion maps), agents can reason about the root causes of physical violations and map them back to the source RTL. For instance, an agent might identify that a specific module’s excessive logic depth is causing negative slack and autonomously rewrite the Verilog to insert pipeline stages. This “RTL-to-GDSII-to-RTL” loop represents the transition from linear automation to circular, self-improving autonomy, a capability unique to the digital design domain where logic and physics are strictly linked via standard cells.

Table 5: Emerging Benchmarks and Datasets for Agentic EDA Research.

Benchmark / Dataset	Focus Area	Reference
<i>ChiPBench</i>	End-to-End Physical Design (PPA)	[57]
<i>Revisiting VerilogEval</i>	RTL Code Generation & Failure Analysis	[35]
<i>CircuitNet 2.0</i>	Large-scale Physical Design Data (14nm)	[58]
<i>HW-NAS-Bench</i>	Hardware-Aware Neural Architecture Search	[59]
<i>OpenLLM-RTL</i>	Open-source RTL Training Data	[39]

4.4 Case Study: Autonomous Flow Orchestration

ChatEDA: Autonomous Tool Orchestration. While ChipChat focused on frontend code generation, *ChatEDA* [22] extends autonomy to the backend RTL-to-GDSII flow. It functions as a hierarchical controller, decomposing high-level user requests (e.g., “Run synthesis and optimize for area”) into executable tool scripts (Tcl/Python). By fine-tuning a LLaMA-2 model (AutoMage) on domain-specific tool manuals, ChatEDA achieves a 98.3% success rate in task planning, demonstrating that agents can effectively orchestrate the complex, multi-stage physical design process without human intervention.

As autonomy expands from isolated tasks to end-to-end flows, it also reshapes the threat model and infrastructure requirements; we therefore next discuss security, trust, and evaluation in agentic EDA systems.

5 Security, Trust & Infrastructure

The transition to Agentic EDA introduces a dual-faceted reality: while autonomous agents significantly enhance productivity, they also expand the attack surface, necessitating a re-evaluation of hardware security and trust. Concurrently, the validation of these agents requires a new generation of infrastructure that moves beyond simple code metrics to holistic performance evaluation.

5.1 The Adversarial Dynamic: Agent vs. Agent Security

The autonomy of EDA agents reshapes hardware security into a dynamic “cat-and-mouse” game. The dual-use nature of LLMs means that the same reasoning capabilities used to optimize designs can be repurposed for malicious intent. This necessitates an equally intelligent defensive posture, creating an adversarial dynamic where offensive and defensive agents co-evolve.

Offensive Agents: Automated Red-Teaming and Stealth. Unlike static, pre-defined hardware trojans, agentic adversaries can adaptively optimize attacks for maximum stealth. Recent studies demonstrate that agents can act as sophisticated “red teams.” *TrojanStego* [60] reveals that agents can function as steganographic carriers, subtly leaking privacy-sensitive design information through code structures without altering functionality. This demonstrates the potential for agents to exploit the complexity of digital logic to hide malicious intent.

Defensive Agents: Cognitive and Intent Analysis. Traditional detection methods based on feature matching often fail against these novel, adaptive attacks. Consequently, defense mechanisms must ascend to a cognitive layer. *TrojanWhisper* [61] leverages the reasoning capabilities of LLMs to perform “intent analysis” on RTL code. By understanding the functional purpose of logic blocks, the defensive agent can distinguish between legitimate corner-case logic and malicious triggers (e.g., a counter that activates a non-functional signal). Advanced multivariate time-series analysis methods [62, 63] can further enhance these defenses by detecting subtle distributional shifts in signal behavior that indicate latent trojan activation.

The Co-evolutionary Loop. Ultimately, the future of hardware security lies in this adversarial loop. Offensive agents serve as automated red teams, generating novel, complex attack vectors that,

crucially, provide the necessary adversarial training data to harden defensive agents against hitherto unknown vulnerabilities.

5.2 Privacy-Preserving Collaboration: Federated Learning

A fundamental paradox in AI-Native EDA is the tension between the need for massive datasets to train robust models and the strict proprietary nature of semiconductor IP. To resolve this, **Federated Learning (FL)** has emerged as a critical enabling technology.

Collaborative Training without IP Leakage. Unlike centralized training which requires pooling sensitive netlists into a single server, FL allows multiple organizations (e.g., design houses and foundries) to collaboratively train a shared global model while keeping their raw data local. As demonstrated in recent frameworks [?], local agents compute gradient updates based on their private designs (e.g., proprietary RISC-V cores) and transmit only these ephemeral gradients to a central aggregator. This ensures that the specific circuit topology never leaves the secure local environment.

Heterogeneous Model Aggregation. A unique challenge in EDA is the heterogeneity of data across different process nodes (e.g., 7nm vs. 28nm). Advanced FL techniques now employ *Domain Adaptation* layers to aggregate knowledge from diverse technology nodes, allowing a model trained on mature nodes to transfer generalizable physical intuition (e.g., congestion patterns) to advanced nodes without violating foundry NDAs.

5.3 Next-Generation Benchmarks and Datasets

To rigorously evaluate Agentic EDA systems, the community is shifting focus from isolated proxy metrics to end-to-end industrial standards.

From Pass Rates to End-to-End PPA. Early benchmarks primarily measured syntactic correctness (e.g., VerilogEval). However, *ChiPBench* [57] exposes a critical misalignment: AI algorithms often optimize intermediate proxy metrics (like wirelength) that do not correlate perfectly with the final Power, Performance, and Area (PPA) after detailed routing. A syntactically correct design that consumes 2x power is useless in industry. Therefore, future benchmarks must be “End-to-End,” evaluating agents based on the final GDSII metrics (WNS/TNS, Area, Power) rather than intermediate proxies.

Open Data Infrastructure and Sovereignty. Data scarcity remains a bottleneck. *CircuitNet 2.0* [58] and its predecessor *CircuitNet* [64] address this by providing large-scale datasets derived from realistic 14nm FinFET designs. The development of such open-source datasets is strategically critical for breaking the “Black Box” monopoly of proprietary EDA vendors and fostering a democratized, autonomous design ecosystem.

These security dynamics and infrastructure constraints complete the picture of what must be solved for practical deployment; we conclude by synthesizing the remaining challenges and outlining the most likely paths toward higher autonomy levels.

6 Conclusion and Future Outlook

The integration of Generative AI into Electronic Design Automation (EDA) represents more than a mere efficiency upgrade; it signifies a fundamental restructuring of the chip design methodology. However, the path from “Automation” to true “Autonomy” is fraught with technical and systemic hurdles. As illustrated in Figure 6, this evolution involves overcoming significant current barriers to realize a future AI-Native autonomous ecosystem.

6.1 Summary of Challenges

Despite the promising results demonstrated by agentic workflows and generative algorithms, several critical barriers remain before widespread industrial adoption can occur, as highlighted by the debate

Current Landscape

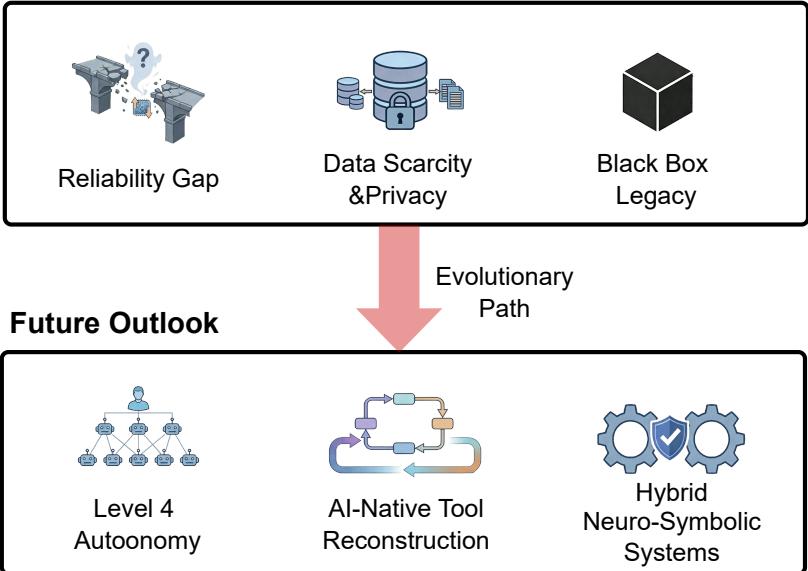


Figure 6: The Evolutionary Path from Current Automation Hurdles to a Future AI-Native Autonomous Ecosystem. The left panel illustrates the primary challenges impeding the transition to true autonomy: reliability issues due to hallucinations, data scarcity from proprietary silos, and the opacity of legacy “black box” tools. The right panel envisions the future AI-Native ecosystem characterized by L4 hierarchical multi-agent autonomy, differentiable and transparent AI-native toolchains, and hybrid neuro-symbolic systems that ensure trustworthy, correct-by-construction designs.

on whether this shift is a “Revolution or Hype” [2]. The left panel of Figure 6 visually summarizes these key hurdles:

- **Reliability and Hallucination:** Unlike software, hardware design has a zero-tolerance policy for errors. The tendency of LLMs to hallucinate syntax or violate strict protocols remains a primary bottleneck. While closed-loop feedback (Section 3.1) mitigates this, achieving “Correct-by-Construction” generation without expensive simulation iterations remains an open challenge.
- **Data Scarcity and Privacy:** High-quality, industrial-grade design data (RTL, netlists, GDSII) is often proprietary and shrouded in secrecy. This data scarcity severely limits the ability to train robust Circuit Foundation Models (CFMs) compared to the vast open-source repositories available for software code.
- **The “Black Box” Legacy:** Current agents are forced to interact with legacy EDA tools that were designed for human interaction, often outputting unstructured logs or opaque binary databases. This “Black Box” nature limits the agent’s ability to reason effectively about the underlying physics of optimization failures.

These three barriers are not isolated; they directly shape what an AI-Native EDA ecosystem must provide. In particular, reliability pressure pushes systems toward formal alignment and verifiable execution; data scarcity pushes training toward privacy-preserving collaboration and stronger open benchmarks; and the black-box toolchain pushes standardization, structured interfaces, and more transparent optimization engines.

Table 6: Proposed Levels of Autonomy for Electronic Design Automation.

Level	Definition	Human Role	Representative System
L0	Manual Design	Operator (Drives tools)	Legacy CAD Tools
L1	AI Assistant	Querier (Asks questions)	General Chatbots (ChatGPT)
L2	AI Copilot	Reviewer (Accepts/Rejects)	VerilogCoder [28], Synopsys.ai Copilot
L3	Task Autonomy	Supervisor (Sets goal for task)	AutoChip [26] (Repair Loop)
L4	Flow Autonomy	Architect (Defines intent)	ChatEDA [22] (RTL-to-GDSII)
L5	Self-Evolving	Observer (Zero intervention)	Future Self-Improving Systems

6.2 Future Trends: Towards AI-Native Autonomy

Looking forward, the evolution of EDA is poised to traverse from the current state of AI-assisted patches to a fully AI-Native ecosystem [3]. The right panel of Figure 6 envisions this future landscape. To clarify the trajectory of this evolution, we propose a standardized taxonomy for EDA autonomy levels, as detailed in Table 6.

As defined in Table 6, the industry is currently transitioning from L2 (Copilot) to L3 and L4 (Task/Flow Autonomy). Future systems will likely feature hierarchical multi-agent societies capable of executing end-to-end flows—from specification to GDSII—with human engineers acting solely as supervisors of intent rather than operators of tools.

To support this transition, explainability becomes paramount. One of the greatest barriers to AI adoption in EDA is the “Black Box” problem. Agentic workflows offer a practical partial remedy via structured execution traces (e.g., tool-call logs, parameter diffs, summarized rationale). Compared with opaque model weights or raw EDA logs, these traces provide a human-readable audit trail (e.g., “increase buffer size because slack is negative”), improving debuggability and accountability in automated decision-making.

Furthermore, instead of merely wrapping LLMs around legacy tools, the next generation of EDA tools will likely be “AI-Native.” A key direction is to make more of the optimization loop differentiable in practice, for example via differentiable surrogates or gradient-aware approximations that connect PPA objectives back to design parameters. This can complement (rather than replace) discrete, heuristic-based solvers by enabling smoother end-to-end optimization and better credit assignment across stages.

Finally, to resolve the reliability crisis, the future belongs to hybrid neuro-symbolic architectures that fuse the pattern-recognition strengths of Deep Learning with the rigorous guarantees of Formal Methods. Together with stronger end-to-end benchmarks (Section 6.3), these developments outline a coherent path from today’s AI-assisted automation toward trustworthy autonomy in chip design.

References

- [1] M. A. Islam, S. Somu, and F. M. F. Aldaihani, “The Rise of Agentic AI: Synthesis of Current Knowledge and Future Research Agenda,” *Global Business and Organizational Excellence*, vol. n/a, no. n/a, eprint: <https://onlinelibrary.wiley.com/doi/pdf/10.1002/joe.70019> TLDR: An antecedent–mechanism–outcome framework linking technological, organizational, and societal enablers to the mechanisms and outcomes of AAI adoption is developed, highlighting AAI as both a driver of business strategy and a potential enabler of organizational excellence and sustainable development. [Online]. Available: <https://onlinelibrary.wiley.com/doi/abs/10.1002/joe.70019>
- [2] Q. Xu, L. Stok, R. Drechsler, X. Wang, G. L. Zhang, and I. L. Markov, “Revolution or Hype? Seeking the Limits of Large Models in Hardware Design,” Sep. 2025, arXiv:2509.04905 [cs]. [Online]. Available: <http://arxiv.org/abs/2509.04905>

- [3] L. Chen, Y. Chen, Z. Chu, W. Fang, T.-Y. Ho, R. Huang, Y. Huang, S. Khan, M. Li, X. Li, Y. Li, Y. Liang, J. Liu, Y. Liu, Y. Lin, G. Luo, Z. Shi, G. Sun, D. Tsaras, R. Wang, Z. Wang, X. Wei, Z. Xie, Q. Xu, C. Xue, J. Yan, J. Yang, B. Yu, M. Yuan, E. F. Y. Young, X. Zeng, H. Zhang, Z. Zhang, Y. Zhao, H.-L. Zhen, Z. Zheng, B. Zhu, K. Zhu, and S. Zou, “The Dawn of AI-Native EDA: Opportunities and Challenges of Large Circuit Models,” *Science China Information Sciences*, vol. 67, no. 10, p. 200402, Oct. 2024, arXiv:2403.07257 [cs] TLDR: This study argues for a paradigm shift from AI4EDA towards AI-rooted EDA from the ground up, integrating AI at the core of the design process, fostering more resilient, efficient, and inventive design methodologies. [Online]. Available: <http://arxiv.org/abs/2403.07257>
- [4] B. Zhang, J. Zhu, and H. Su, “Toward third-generation artificial intelligence,” *Science China Information Sciences*, vol. 66, no. 2, p. 121101, 2023.
- [5] W. Fang, J. Wang, Y. Lu, S. Liu, Y. Wu, Y. Ma, and Z. Xie, “A Survey of Circuit Foundation Model: Foundation AI Models for VLSI Circuit Design and EDA,” Mar. 2025, arXiv:2504.03711 [cs]. [Online]. Available: <http://arxiv.org/abs/2504.03711>
- [6] Z. He, Y. Pu, H. Wu, T. Qiu, and B. Yu, “Large Language Models for EDA: Future or Mirage?” *ACM Trans. Des. Autom. Electron. Syst.*, vol. 30, no. 6, pp. 90:1–90:53, Oct. 2025, tLDR: A comprehensive, evidence-based perspective on the role of LLMs in shaping the future of EDA is provided by organizing existing research into four critical domains of EDA—code generation, verification and debugging, knowledge representation and retrieval, and optimization/modeling. [Online]. Available: <https://dl.acm.org/doi/10.1145/3736167>
- [7] W. Fang, S. Liu, J. Wang, and Z. Xie, “CircuitFusion: Multimodal Circuit Representation Learning for Agile Chip Design,” May 2025, arXiv:2505.02168 [cs] TLDR: This paper introduces CircuitFusion, the first multimodal and implementation-aware circuit encoder that consistently outperforms the SOTA supervised method specifically developed for every single task, demonstrating its generalizability and ability to learn circuits’ inherent properties. [Online]. Available: <http://arxiv.org/abs/2505.02168>
- [8] S. Li, Z. Liu, Z. Zang, D. Wu, Z. Chen, and S. Z. Li, “Genurl: A general framework for unsupervised representation learning,” *IEEE Transactions on Neural Networks and Learning Systems*, vol. 36, no. 1, pp. 286–298, 2024.
- [9] L. Chen, Y. Yang, B. Yu, D. Z. Pan *et al.*, “Large circuit models: Opportunities and challenges,” *arXiv preprint arXiv:2311.00926*, 2023, also associated with discussions on AI-Native EDA foundations.
- [10] C. Tian *et al.*, “A survey for electronic design automation based on graph neural network,” *Integration*, vol. 94, p. 102091, 2024.
- [11] Z. Zheng, S. Huang, J. Zhong, Z. Shi, G. Dai, N. Xu, and Q. Xu, “DeepGate4: Efficient and Effective Representation Learning for Circuit Design at Scale,” May 2025, arXiv:2502.01681 [cs] TLDR: DeepGate4 is introduced, a scalable and efficient graph transformer specifically designed for large-scale circuits that significantly surpasses state-of-the-art methods and demonstrates the potential of DeepGate4 to handle complex EDA tasks while offering superior scalability and efficiency. [Online]. Available: <http://arxiv.org/abs/2502.01681>
- [12] Z. Luo, T. S. Hy, P. Tabaghi, M. Defferrard, E. Rezaei, R. M. Carey, R. Davis, R. Jain, and Y. Wang, “DE-HNN: An effective neural model for Circuit Netlist representation,” in *Proceedings of The 27th International Conference on Artificial Intelligence and Statistics*. PMLR, Apr. 2024, pp. 4258–4266, iSSN: 2640-3498. [Online]. Available: <https://proceedings.mlr.press/v238/luo24a.html>

- [13] Z. Zang, S. Li, D. Wu, J. Guo, Y. Xu, and S. Z. Li, “Deep manifold embedding of attributed graphs,” *Neurocomputing*, vol. 514, pp. 83–93, 2022.
- [14] B. Hu, Z. Zang, J. Xia, L. Wu, C. Tan, and S. Z. Li, “Deep manifold graph auto-encoder for attributed graph embedding,” in *ICASSP 2023-2023 IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP)*. IEEE, 2023, pp. 1–5.
- [15] S. Li, H. Lin, Z. Zang, L. Wu, J. Xia, and S. Z. Li, “Invertible manifold learning for dimension reduction,” in *ECML PKDD*, vol. 21, no. 13–17. Springer International Publishing, 2021, pp. 713–728.
- [16] S. Luan, C. Hua, Q. Lu, L. Ma, L. Wu, X. Wang, M. Xu, X.-W. Chang, Z. Zang, D. Precup *et al.*, “The heterophilic graph learning handbook: Benchmarks, models, theoretical analysis, applications and challenges,” *arXiv preprint arXiv:2407.09618*, 2024.
- [17] L. Wu, Z. Liu, Z. Zang, J. Xia, S. Li, and S. Z. Li, “Deep clustering and representation learning that preserves geometric structures,” 2020.
- [18] Y. Xu, Z. Zang, J. Xia, C. Tan, Y. Geng, and S. Z. Li, “Structure-preserving visualization for single-cell rna-seq profiles using deep manifold transformation with batch-correction,” *Communications Biology*, vol. 6, no. 1, p. 369, 2023.
- [19] Z. Zang, S. Cheng, H. Xia, L. Li, Y. Sun, Y. Xu, L. Shang, B. Sun, and S. Z. Li, “Dmt-ev: An explainable deep network for dimension reduction,” *IEEE Transactions on Visualization and Computer Graphics*, vol. 30, no. 3, pp. 1710–1727, 2024.
- [20] H. Wu, Z. He, X. Zhang, X. Yao, S. Zheng, H. Zheng, and B. Yu, “ChatEDA: A Large Language Model Powered Autonomous Agent for EDA,” *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, vol. 43, no. 10, pp. 3184–3197, Oct. 2024, tLDR: ChatEDA, an autonomous agent for EDA empowered by an LLM, AutoMage, complemented by EDA tools serving as executors is introduced, streamlines the design flow from the register-transfer level (RTL) to the graphic data system version II (GDSII) by effectively managing task decomposition, script generation, and task execution. [Online]. Available: <https://ieeexplore.ieee.org/document/10485372>
- [21] X. Wang, B. Han, Z. Tai, J. Tian, Y. Wang, J. Yan, and Y. Tian, “New Interaction Paradigm for Complex EDA Software Leveraging GPT,” Aug. 2025, arXiv:2307.14740 [cs] tLDR: Preliminary results suggest that SmartonAI significantly reduces onboarding time and enhances productivity, representing a promising step toward generalizable AI-assisted interaction paradigms for complex software systems. [Online]. Available: <http://arxiv.org/abs/2307.14740>
- [22] Z. He, H. Wu, X. Zhang, X. Yao, S. Zheng, H. Zheng, and B. Yu, “ChatEDA: A Large Language Model Powered Autonomous Agent for EDA,” *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, vol. 43, no. 10, pp. 3184–3197, Oct. 2024, arXiv:2308.10204 [cs] tLDR: ChatEDA, an autonomous agent for EDA empowered by an LLM, AutoMage, complemented by EDA tools serving as executors is introduced, streamlines the design flow from the register-transfer level (RTL) to the graphic data system version II (GDSII) by effectively managing task decomposition, script generation, and task execution. [Online]. Available: <http://arxiv.org/abs/2308.10204>
- [23] M. Ahmadzadeh, K. Chen, and G. Gielen, “AnaFlow: Agentic LLM-based Workflow for Reasoning-Driven Explainable and Sample-Efficient Analog Circuit Sizing,” Nov. 2025, arXiv:2511.03697 [cs] tLDR: A novel agentic AI framework for sample-efficient and explainable analog circuit sizing is presented and is able to complete the sizing task fully automatically, differently from pure Bayesian optimization and reinforcement learning approaches. [Online]. Available: <http://arxiv.org/abs/2511.03697>

- [24] S. Ma, J. Huang, B. Yang, F. Zhang, J. Wu, Y. Shen, G. Fan, Z. Zhang, and Z. Zang, “Medla: A logic-driven multi-agent framework for complex medical reasoning with large language models,” *AAAI 26*, 2026.
- [25] S. Jiang, M. Xie, F. Y. Chen, J. Ma, and J. Luo, “Intelligent Design 4.0: Paradigm Evolution Toward the Agentic AI Era,” *Journal of Computing and Information Science in Engineering*, vol. 25, no. 12, p. 120808, Dec. 2025, arXiv:2506.09755 [cs]. [Online]. Available: <http://arxiv.org/abs/2506.09755>
- [26] S. Thakur, J. Blocklove, H. Pearce, B. Tan, S. Garg, and R. Karri, “AutoChip: Automating HDL generation using LLM feedback,” 2023. [Online]. Available: <https://arxiv.org/abs/2311.04887>
- [27] H. Huang, Z. Lin, Z. Wang, X. Chen, K. Ding, and J. Zhao, “Towards LLM-Powered Verilog RTL Assistant: Self-Verification and Self-Correction,” May 2024, arXiv:2406.00115 [cs] tLDR: VeriAssist, an LLM-powered programming assistant for Verilog RTL design workflow, significantly improves both syntax and functionality correctness over existing LLM implementations, thus minimizing human intervention and making RTL design more accessible to novice designers. [Online]. Available: <http://arxiv.org/abs/2406.00115>
- [28] C.-T. Ho, H. Ren, and B. Khailany, “VerilogCoder: Autonomous Verilog Coding Agents with Graph-based Planning and Abstract Syntax Tree (AST)-based Waveform Tracing Tool,” *Proceedings of the AAAI Conference on Artificial Intelligence*, vol. 39, no. 1, pp. 300–307, Apr. 2025. [Online]. Available: <https://ojs.aaai.org/index.php/AAAI/article/view/32007>
- [29] M. Liu, Y.-D. Tsai, W. Zhou, and H. Ren, “CraftRTL: High-quality Synthetic Data Generation for Verilog Code Models with Correct-by-Construction Non-Textual Representations and Targeted Code Repair,” Feb. 2025, arXiv:2409.12993 [cs] tLDR: This paper presents an analysis of fine-tuned LLMs on Verilog coding, with synthetic data from prior methods and introduces an automated framework that generates error reports from various model checkpoints and injects these errors into open-source code to create targeted code repair data. [Online]. Available: <http://arxiv.org/abs/2409.12993>
- [30] Y. Yang, F. Teng, P. Liu, M. Qi, C. Lv, J. Li, X. Zhang, and Z. He, “HaVen: Hallucination-mitigated LLM for Verilog code generation aligned with HDL engineers,” in *Proceedings of the 2025 Design, Automation & Test in Europe Conference (DATE)*, 2025, arXiv:2501.04908.
- [31] N. Zhang, C. Deng, J. M. Kuehn, C.-T. Ho, C. Yu, Z. Zhang, and H. Ren, “ASPEN: LLM-guided E-graph rewriting for RTL datapath optimization,” in *2025 ACM/IEEE Workshop on Machine Learning for CAD (MLCAD)*, 2025.
- [32] A. Kumar, D. N. Gadde, K. K. Radhakrishna, and D. Lettnin, “Saarthi: The first AI formal verification engineer,” in *DVCon U.S. 2025*, 2025, arXiv:2502.16662.
- [33] C.-C. Chang, C.-T. Ho, Y. Li, Y. Chen, and H. Ren, “DRC-Coder: Automated DRC Checker Code Generation Using LLM Autonomous Agent,” in *Proceedings of the 2025 International Symposium on Physical Design*, ser. ISPD ’25. New York, NY, USA: Association for Computing Machinery, Mar. 2025, pp. 143–151, tLDR: DRC-Coder is presented, a multi-agent framework with vision capabilities for automated DRC code generation that can generate code for each design rule within four minutes on average, which significantly accelerates technology advancement and reduces engineering costs. [Online]. Available: <https://dl.acm.org/doi/10.1145/3698364.3705347>
- [34] S. Thakur, B. Ahmad, H. Pearce, B. Tan, B. Dolan-Gavitt, R. Karri, and S. Garg, “VeriGen: A Large Language Model for Verilog Code Generation,” *ACM Trans. Des. Autom. Electron. Syst.*, vol. 29, no. 3, pp. 46:1–46:31, Apr. 2024, tLDR: A fine-tuned open-source CodeGen-16B model demonstrates a 41% improvement in generating syntactically correct Verilog code

across various problem categories compared to its pre-trained counterpart, highlighting the potential of smaller, in-house LLMs in hardware design automation. [Online]. Available: <https://dl.acm.org/doi/10.1145/3643681>

- [35] N. Pinckney, C. Batten, M. Liu, H. Ren, and B. Khailany, “Revisiting VerilogEval: A Year of Improvements in Large-Language Models for Hardware Code Generation,” *ACM Trans. Des. Autom. Electron. Syst.*, vol. 30, no. 6, pp. 91:1–91:20, Oct. 2025, tLDR: This work evaluates new commercial and open models since VerilogEval’s original release—including GPT-4o, GPT-4 Turbo, Llama3.1 (8B/70B/405B), Llama3 70B, Mistral Large, DeepSeek Coder, CodeGemma 7B, and RTL-Coder—against an improved VerilogEval benchmark suite and enhances VerilogEval’s infrastructure. [Online]. Available: <https://dl.acm.org/doi/10.1145/3718088>
- [36] K. Chang, Y. Wang, H. Ren, M. Wang, S. Liang, Y. Han, H. Li, and X. Li, “ChipGPT: How far are we from natural language hardware design,” Oct. 2025, arXiv:2305.14019 [cs] tLDR: This work attempts to demonstrate an automated design environment that explores LLMs to generate hardware logic designs from natural language specifications without retraining or finetuning, and shows broader design optimization space compared to prior work and native LLMs alone. [Online]. Available: <http://arxiv.org/abs/2305.14019>
- [37] R. Baartmans, A. Ensinger, V. Agostinelli, and L. Chen, “ML For Hardware Design Interpretability: Challenges and Opportunities,” Apr. 2025, arXiv:2504.08852 [cs] tLDR: This paper examines how design interpretability, particularly in RTL-to-NL tasks, influences the efficiency of the hardware design process, thereby accelerating the hardware design process and meeting the increasing demand for custom hardware accelerators in machine learning and beyond. [Online]. Available: <http://arxiv.org/abs/2504.08852>
- [38] Z. Zang, H. Luo, K. Wang, P. Zhang, F. Wang, S. Li, and Y. You, “Diffaug: Enhance unsupervised contrastive learning with domain-knowledge-free diffusion-based data augmentation,” in *International Conference on Machine Learning (ICML24)*, 2024.
- [39] S. Liu, Y. Lu, W. Fang, M. Li, and Z. Xie, “OpenLLM-RTL: Open Dataset and Benchmark for LLM-Aided Design RTL Generation,” Mar. 2025, arXiv:2503.15112 [cs]. [Online]. Available: <http://arxiv.org/abs/2503.15112>
- [40] ———, “OpenLLM-RTL: Open Dataset and Benchmark for LLM-Aided Design RTL Generation,” in *Proceedings of the 43rd IEEE/ACM International Conference on Computer-Aided Design*. Newark Liberty International Airport Marriott New York NY USA: ACM, Oct. 2024, pp. 1–9, tLDR: Three studies are integrated into one framework, providing off-the-shelf support for the development and evaluation of LLMs for RTL code generation and verification, indicating that LLM performance can be boosted by enlarging the training dataset, improving data quality, and improving the training scheme. [Online]. Available: <https://dl.acm.org/doi/10.1145/3676536.3697118>
- [41] A. Gupta, B. Mali, and C. Karfa, “Sangam: Systemverilog assertion generation via monte carlo tree self-refine,” *arXiv preprint arXiv:2506.13983*, 2025.
- [42] C. R. Pamnani, “AI-Driven Automation for Digital Hardware Design: A Multi-Agent Generative Approach,” in *Proceedings of the 2025 4th International Conference on Frontiers of Artificial Intelligence and Machine Learning*, ser. FAIML ’25. New York, NY, USA: Association for Computing Machinery, Aug. 2025, pp. 26–30, tLDR: This study introduces a novel AI-driven framework that leverages multi-agent collaboration and generative modeling to optimize the hardware design process, and autonomously generates and refines hardware description language (HDL) code, improving design accuracy and performance. [Online]. Available: <https://dl.acm.org/doi/10.1145/3748382.3748388>

- [43] Z. Aref, R. Suvarna, B. Hughes, S. Srinivasan, and N. B. Mandayam, “Advanced Reinforcement Learning Algorithms to Optimize Design Verification,” in *Proceedings of the 61st ACM/IEEE Design Automation Conference*, ser. DAC ’24. New York, NY, USA: Association for Computing Machinery, Nov. 2024, pp. 1–6, tLDR: This paper proposes a deep deterministic policy gradient (DDPG) algorithm combined with prioritized experience replay (PER) to determine the stimulus settings that result in the highest average FIFO depth in a modified exclusive shared invalid (MESI) cache controller architecture. [Online]. Available: <https://dl.acm.org/doi/10.1145/3649329.3657365>
- [44] Y. Zhao, D. Wu *et al.*, “PRO-V-R1: Reasoning enhanced programming agent for RTL verification,” *arXiv preprint arXiv:2506.12200*, 2025. [Online]. Available: <https://arxiv.org/abs/2506.12200>
- [45] C. Bai, G. Hamad *et al.*, “FVDebug: An LLM-driven debugging assistant for automated root cause analysis of formal verification failures,” *arXiv preprint arXiv:2510.15906*, 2025. [Online]. Available: <https://arxiv.org/abs/2510.15906>
- [46] Y. Gao, D. Luo, C. Bai, B. Yu, H. Geng, Q. Sun, and C. Zhuo, “Is Vanilla Bayesian Optimization Enough for High-Dimensional Architecture Design Optimization?” in *Proceedings of the 43rd IEEE/ACM International Conference on Computer-Aided Design*. New York, NY, USA: Association for Computing Machinery, Apr. 2025, no. 141, pp. 1–9. [Online]. Available: <https://doi.org/10.1145/3676536.3676746>
- [47] J. Blocklove, S. Garg, R. Karri, and H. Pearce, “Chip-chat: Challenges and opportunities in conversational hardware design,” in *2023 ACM/IEEE 5th Workshop on Machine Learning for CAD (MLCAD)*, 2023, pp. 1–6.
- [48] Y. Lai, Y. Mu, and P. Luo, “MaskPlace: Fast Chip Placement via Reinforced Visual Representation Learning,” *Advances in Neural Information Processing Systems*, vol. 35, pp. 24 019–24 030, Dec. 2022. [Online]. Available: https://proceedings.neurips.cc/paper_files/paper/2022/hash/97c8a8eb0e5231d107d0da51b79e09cb-Abstract-Conference.html
- [49] V. Lee, M. Nguyen, L. Elzeiny, C. Deng, P. Abbeel, and J. Wawrzynek, “Chip Placement with Diffusion Models,” Jun. 2025, arXiv:2407.12282 [cs]. [Online]. Available: <http://arxiv.org/abs/2407.12282>
- [50] M. Ning, M. Li, J. Su, H. Jia, L. Liu, M. Beneš, W. Chen, A. A. Salah, and I. O. Ertugrul, “DCTdiff: Intriguing Properties of Image Generative Modeling in the DCT Space,” May 2025, arXiv:2412.15032 [cs] tLDR: DCTdiff, an end-to-end diffusion generative paradigm that efficiently models images in the discrete cosine transform (DCT) space, is introduced and a theoretical proof of why ‘image diffusion can be seen as spectral autoregression’, bridging the gap between diffusion and autoregressive models is provided. [Online]. Available: <http://arxiv.org/abs/2412.15032>
- [51] Y. Hou, H. Ye, S. Yang, Y. Zhang, S. Xu, and G. Song, “TransPlace: Transferable Circuit Global Placement via Graph Neural Network,” Mar. 2025, arXiv:2501.05667 [cs]. [Online]. Available: <http://arxiv.org/abs/2501.05667>
- [52] A. Ghose, A. B. Kahng, S. Kundu, and Z. Wang, “ORFS-agent: Tool-Using Agents for Chip Design Optimization,” Aug. 2025, arXiv:2506.08332 [cs]. [Online]. Available: <http://arxiv.org/abs/2506.08332>
- [53] A. B. Kahng, “Solvers, Engines, Tools and Flows: The Next Wave for AI/ML in Physical Design,” in *Proceedings of the 2024 International Symposium on Physical Design*, ser. ISPD ’24. New York, NY, USA: Association for Computing Machinery, Mar. 2024, pp. 117–124, tLDR: It has been six years since an ISPD-2018 invited talk on ”Machine Learning Applications in

- Physical Design”, and there is now clearer understanding of where AI/ML can and cannot move the needle in physical design, as well as some of the difficult blockers and technical challenges that lie ahead. [Online]. Available: <https://dl.acm.org/doi/10.1145/3626184.3635277>
- [54] X. Luo, Y. Li *et al.*, “Tscompiler: efficient compilation framework for dynamic-shape models,” *IEEE Transactions on Parallel and Distributed Systems (TPDS)*, 2024, please verify the specific issue/volume as it is a recent publication.
- [55] B.-Y. Wu, R. Liang, G. Pradipta, A. Agnesina, H. Ren, and V. A. Chhabria, “2024 ICCAD CAD Contest Problem C: Scalable Logic Gate Sizing Using ML Techniques and GPU Acceleration,” in *Proceedings of the 43rd IEEE/ACM International Conference on Computer-Aided Design*, ser. ICCAD ’24. New York, NY, USA: Association for Computing Machinery, Apr. 2025, pp. 1–5, tLDR: This contest aims to advance logic gate sizing and push the boundaries of PPA improvement through innovative EDA tools that leverage machine learning and GPU acceleration, and leverages the open-source EDA tool OpenROAD and ML-friendly data representation format, CircuitOps. [Online]. Available: <https://dl.acm.org/doi/10.1145/3676536.3689912>
- [56] K. Min, K. Cho, J. Jang, and S. Kang, “Revolution: An evolutionary framework for rtl generation driven by large language models,” *arXiv preprint arXiv:2510.21407*, 2025, accepted for publication at the 2026 Asia and South Pacific Design Automation Conference (ASP-DAC).
- [57] Z. Wang, Z. Geng, Z. Tu, J. Wang, Y. Qian, Z. Xu, Z. Liu, S. Xu, Z. Tang, S. Kai, M. Yuan, J. Hao, B. Li, Y. Zhang, and F. Wu, “Benchmarking End-To-End Performance of AI-Based Chip Placement Algorithms,” Dec. 2024, arXiv:2407.15026 [cs] tLDR: ChiPBench is a comprehensive benchmark specifically designed to evaluate the effectiveness of existing AI-based chip placement algorithms in improving final design PPA metrics and shows that even if intermediate metric of a single-point algorithm is dominant, while the final PPA results are unsatisfactory. [Online]. Available: <http://arxiv.org/abs/2407.15026>
- [58] X. Jiang, Z. Chai, Y. Zhao, Y. Lin, R. Wang, and R. Huang, “CircuitNet 2.0: An Advanced Dataset for Promoting Machine Learning Innovations in Realistic Chip Design Environment,” Oct. 2023. [Online]. Available: <https://openreview.net/forum?id=nMFSUjxMII>
- [59] C. Li, Z. Yu, Y. Fu, Y. Zhang, Y. Zhao, H. You, Q. Yu, Y. Wang, and Y. C. Lin, “HW-NAS-Bench:Hardware-Aware Neural Architecture Search Benchmark,” Mar. 2025, arXiv:2103.10584 [cs]. [Online]. Available: <http://arxiv.org/abs/2103.10584>
- [60] T. Meier *et al.*, “TrojanStego: Your language model can secretly be a steganographic privacy leaking agent,” 2025, accessed: 2025-12-05. [Online]. Available: <https://gipplab.uni-goettingen.de/wp-content/papercite-data/pdf/meier2025b.pdf>
- [61] Unknown, “TrojanWhisper: LLMs detect hardware trojans in RTL,” Scribd Document, 2025, context-aware hardware trojan detection using Large Language Models. [Online]. Available: <https://www.scribd.com/document/809542994/TrojanWhisper>
- [62] H. Liu, X. Qiu, Y. Shi, M. Xu, Z. Zang, and Z. Lei, “Deep multimanifold transformation-based multivariate time series fault detection,” *IEEE Transactions on Neural Networks and Learning Systems*, 2025.
- [63] H. Liu, X. Qiu, Y. Shi, and Z. Zang, “Usd: Unsupervised soft contrastive learning for fault detection in multivariate time series,” in *ICASSP 2025-2025 IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP)*. IEEE, 2025, pp. 1–5.
- [64] Z. Chai, Y. Zhao, Y. Lin, W. Liu, R. Wang, and R. Huang, “Circuitnet: An open-source dataset for machine learning applications in electronic design automation (eda),” *Science China Information Sciences*, vol. 66, no. 12, p. 229101, 2023.