Special Research Session: Progress and Roadmap Toward an Automated EDA R&D Engineer

Amur Ghose
University of California San Diego
aghose@ucsd.edu

Andrew B. Kahng
University of California San Diego
abk@ucsd.edu

Bodhisatta Pramanik
University of California San Diego
bopramanik@ucsd.edu

Abstract

An EDA R&D engineer operates across three coupled *modes* of work: producing physical design (PD) artifacts with expert judgment, developing new algorithms within production tools, and maintaining the underlying tool codebases. Recent agentic AI systems built on large language models (LLMs) have begun to automate these modes individually. This paper examines what is required to compose such capabilities into an automated EDA R&D engineer. We extend the agentic EDA taxonomy of [4, 18] by introducing a *design-task* tier, and by separating *algorithm-level* automation from *codebase-level* automation within the former code-level tier. We ground this taxonomy in three existence proofs: agentic macro placement, evolutionary improvement of OpenROAD placement code, and repository-scale OpenROAD optimization under closed-loop PD feedback with optional machine-checked correctness gates. The limitations of these systems identify bottlenecks in evaluation, orchestration, memory, and verification, suggesting that the remaining path toward automated EDA R&D is primarily a systems challenge.

ACM Reference Format:
Amur Ghose, Andrew B. Kahng, and Bodhisatta Pramanik. 2026. Special Research Session: Progress and Roadmap Toward an Automated EDA R&D Engineer. In *63rd ACM/IEEE Design Automation Conference (DAC '26), July 26–29, 2026, Long Beach, CA, USA.* ACM, New York, NY, USA, 4 pages. https://doi.org/10.1145/3770743.3815396

1 Introduction

Recent progress in LLMs has shifted the discussion in chip design automation from whether models can answer EDA questions to whether agents can close the loop on real design tasks. This distinction is important. A language model primarily produces text. An agent operates in a mutable environment: it selects tools, edits artifacts, interprets logs, recovers from failures, and is evaluated by the effect of its actions on a task. In software engineering, the transition from base models to reliable agents has been driven not only by model scaling, but also by executable environments and tool-use benchmarks, including ReAct-style tool use [19], SWE-bench [8], and Terminal-Bench [13]. Physical design (PD) is approaching a similar inflection point.

A human EDA R&D engineer works across three coupled modes. First, the engineer produces design artifacts, such as floorplans and macro placements, using expert judgment. Second, the engineer develops new algorithms within EDA tools. Third, the engineer maintains and extends the underlying tool codebase. In current practice, the first mode is often partially outsourced: a human R&D engineer can rely on a design team, or on a previously released reference floorplan, to provide a sensible starting point against which a new algorithm is evaluated. An *automated* EDA R&D engineer does not have this option. Closing the R&D loop requires the system itself to bring a design to a *reasonable* baseline before the algorithm's contribution becomes measurable. Design-task capability is therefore not an auxiliary skill but a prerequisite for end-to-end validation. Automating any one of these modes is difficult. Automating all three, and composing them into a closed loop that can formulate, implement, evaluate, and refine PD ideas, is the goal of an *automated EDA R&D engineer*. Our conjecture is that progress toward this goal is now limited less by raw foundation-model capability than by the surrounding systems stack: interfaces, orchestration, memory, verification, and experiment management.

Recent work makes this shift visible. Domain-adapted LLMs and retrieval-grounded assistants have reduced the cost of scripting, debugging, and documentation tasks [10, 12]. Natural-language agents now orchestrate substantial portions of PD flows end-to-end [5, 17]. In parallel, LLM-driven code search has advanced from self-contained program synthesis [15, 16] to targeted modification of production placers [20], and more recently to repository-scale edits of open EDA toolchains [3, 7, 21]. Recent surveys [4, 18] organize this landscape into two main categories: *flow-level* agents, which operate through tool interfaces and scripts, and *code-level* agents, which modify EDA implementations. This taxonomy captures the first wave of progress, but is incomplete in two aspects. (i) It conflates distinct forms of code modification, ranging from localized algorithmic edits to cross-repository engineering. (ii) It omits a central mode of PD practice: producing the design artifact itself.

This paper makes three contributions. (i) We refine the existing taxonomy of agentic EDA systems by separating code-level automation into *algorithm-level* and *codebase-level* sub-tiers, and by adding a *design-task* tier for agents that directly produce design artifacts. (ii) We ground this taxonomy in three recent example systems: agentic macro placement, evolutionary improvement of OpenROAD placement code, and repository-scale OpenROAD optimization. (iii) We propose that the next bottlenecks are primarily systems problems: machine-readable interfaces, provenance-aware memory, experiment registration, cross-tier orchestration, scalable verification, and trustworthy evaluation under noisy PD objectives.

The rest of the paper is organized as follows. Section 2 presents the three-tier taxonomy of agents for EDA R&D automation. Section 3 describes three recent agentic systems that instantiate the tiers. Section 4 presents a roadmap toward an autonomous EDA R&D engineer, and Section 5 concludes the paper.

2 A Taxonomy of Agentic EDA R&D Automation

We organize agentic AI systems for EDA R&D into three tiers, distinguished by the *artifact* the agent modifies or produces. These tiers are not levels of autonomy. Rather, they identify the object of agent action: a flow invocation, a tool implementation, or a design artifact.

- **Tier 1 — Flow-level agents** treat EDA tools as black boxes and optimize their orchestration, configuration, and parameterization. ChatEDA [17] translates natural-language intent into executable flow scripts. ORAssistant [10] retrieves over OpenROAD documentation to answer tool questions and generate scripts. ORFS-agent [5] uses an LLM with tool-calling to tune OpenROAD-flow-scripts (ORFS) parameters, reaching iso-QoR in roughly 40% fewer iterations than Bayesian optimization. Tier 1 agents do not change the computation performed by the underlying tools; they change how those tools are invoked. This tier has been well-surveyed in [4].

- **Tier 2 — Code-level agents** modify the source code of EDA tools. The artifact they produce is a repository diff, rather than a script, command sequence, or parameter vector. Within this tier, we distinguish two sub-tiers that were not identified in the two-tier taxonomy of [4]. The first is *algorithm-level* evolution, in which an agent discovers or modifies algorithmic logic within a localized module, e.g., a placement cost function. The second is *codebase-level* improvement, in which an agent reasons across modules under closed-loop QoR feedback and may attach machine-checkable contracts to its edits. This distinction is important because the two sub-tiers differ in scope, failure modes, verification requirements, and the LLM capabilities they stress (see Section 3).
- **Tier 3 — Design-task agents** do not modify the EDA tool. Instead, they use tools to produce design artifacts, such as floorplans, macro placements, or pin assignments. These tasks require the kind of expert judgment that human designers provide and that classical automation often fails to reproduce consistently. This tier does not appear in [4]; however, it is essential to EDA R&D automation because human engineers spend substantial effort producing and refining design artifacts. It also stresses capabilities that are largely orthogonal to flow orchestration and code editing (multimodal spatial reasoning, principle-based decomposition, etc.). Tier 3 is therefore the tier most directly comparable to human design output, and the tier where "human-quality" can be evaluated most explicitly.

A complete EDA R&D workflow often spans all three tiers. A researcher may read the literature, formulate a new algorithmic idea (Tier 2 algorithm-level), implement it in the tool repository (Tier 2 codebase-level), expose it through a flow or script interface (Tier 1), and evaluate it on designs whose outcomes may depend on high-quality floorplans or macro placements (Tier 3). No current agent performs this workflow end-to-end. In the following, we examine recent progress in these tiers and identify systems challenges that remain before these capabilities can be composed.

3 Recent Progress: Three Existence Proofs

This section grounds the proposed taxonomy in three existence proofs spanning the newly emphasized design-task tier and the two distinct sub-tiers of code-level automation; flow-level agents have been comparatively well covered in prior work [4, 18]. Our goal is not to provide a comprehensive survey. Instead, we identify representative existence proofs that expose the current capabilities and limitations of each tier.[1] For each example system, we ask four questions: what artifact the agent modifies or produces, what closed loop connects proposal to evaluation, what evidence supports improvement, and what limitations remain before the capability can be integrated into a broader automated R&D workflow.

3.1 Macro placement as a design-task agent

Macro placement is a canonical Tier 3 problem. It is NP-hard, strongly coupled to downstream PPA, and still routinely refined by expert designers after automated tools produce legal placements. Classical analytical and hierarchy-aware placers [2, 9], RL-based methods [14], and analytical frameworks [1] can achieve competitive proxy metrics. However, to date they do not consistently reproduce the regularity and polish associated with expert floorplans, in terms of boundary snapping, coherent macro alignment, whitespace organization, and pin-aware orientation. These forms of design judgment have historically remained difficult to automate.

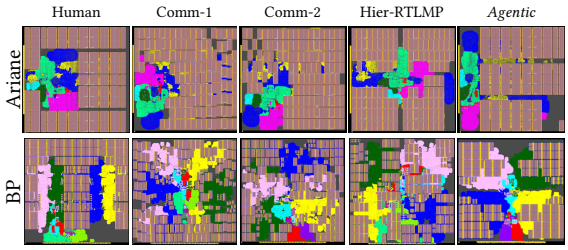

Figure 1: Macro placement comparisons on NG45 designs across human, commercial, open-source, and agentic methods. The agentic placements qualitatively exhibit stronger boundary alignment, whitespace organization, and macro regularity than the automated baselines.

Table 1: PPA comparison on NG45 enablement. rWL: routed wirelength (mm), Pwr: total power (mW), WNS: worst negative slack (ps), TNS: total negative slack (ns). Best per design in bold. Comm-(1,2) denote two commercial baselines.

Design	Method	rWL	Pwr	WNS	TNS
Ariane	Human	4681	832	-88	-46.8
	Comm-1	4057	852	-154	-196.5
	Comm-2	**3994**	**830**	-144	-127.9
	Hier-RTLMP	5297	836	-165	-223.4
	Agentic	4424	832	**-81**	**-32.7**
BP	Human	25916	4470	-97	-321.9
	Comm-1	**23144**	4429	-144	-356.2
	Comm-2	24080	4447	-182	-842.0
	Hier-RTLMP	28866	4512	-150	-456.6
	Agentic	26987	4479	**-88**	**-139.7**

Agentic principles as scaffolding. We formulate macro placement as a structured multi-agent reasoning problem. Unlike RL-based placers that require per-design reward optimization or supervised approaches that would demand curated corpora of expert floorplans (which are largely unavailable due to IP constraints), the agentic system draws on two complementary sources of design knowledge: the pretrained spatial and visual priors of a vision-language model, and explicitly authored floorplanning principles. The system decomposes the macro placement task into explicit floorplanning decisions guided by domain principles. Expert knowledge, including boundary snapping, macro alignment, IO keepout margins, and pin-aware orientation rules, is encoded as structured directives rather than learned from labeled data. This makes the system interpretable, editable by domain experts, and portable across designs without task-specific retraining or dataset curation. The results suggest a favorable timing/structure tradeoff: the agentic placer achieves the best WNS/TNS in Table 1, remains competitive in routed wirelength and power, and produces layouts with stronger human-like spatial structure (Figure 1) than other baselines.

Tier 3 implication. These results show that with current tooling, aspects of expert floorplanning judgment can be operationalized without per-design training. Design tasks expressible through geometric constraints, principle compliance, and structured critique are today natural candidates for multimodal agentic pipelines; tasks whose objectives are difficult to specify or audit remain open, but may become tractable as evaluation infrastructure (surrogates, multi-fidelity proxies) and verification methods mature (Section 4).

3.2 Code evolution as an algorithm-level agent

Improving a placement algorithm is traditionally a slow and expensive process. An engineer must understand a large C++ codebase,

[1] These examples should be read as representative case studies only: each demonstrates feasibility at one tier, while also exposing systems-level limitations.

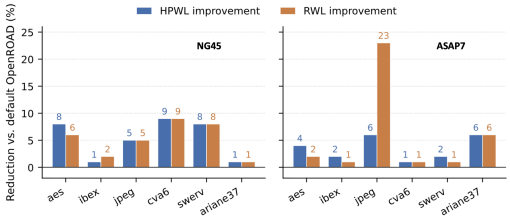

Figure 2: Agent-evolved placement results: HPWL and routed-wirelength (RWL) reductions relative to default OpenROAD.

work within a production build system, and evaluate QoR across multiple designs and technology nodes before determining whether an idea is useful. LLM-guided code-search frameworks [15, 16] have shown that such loops can be automated for small, self-contained programs. EvoPlace [11] extended this paradigm to interpreted Python optimizer code in DREAMPlace. However, it remained unclear whether similar agentic search could scale to compiled C++ inside a production RTL-to-GDS flow, where candidate changes must build successfully, interact correctly with existing modules, and be evaluated through downstream PPA [7].

Agentic principles as scaffolding. We formulate placement improvement as evolutionary search over edits to OpenROAD placement modules. LLM coding agents propose source-level mutations, compile each candidate in isolation, and evaluate successful builds under closed-loop placement feedback. Making this search tractable in a large compiled codebase requires three ingredients:

- *Structured grounding.* The agent must be grounded in previous literature, pseudocode abstractions of performance-critical functions, catalogs of tunable parameters with reasonable ranges, and lightweight profiling data. Without such grounding, code search tends to produce implausible or low-value edits.
- *A spectrum of mutation operators.* The search must span conservative parameter perturbations, localized logic changes, and more aggressive algorithmic rewrites. The operator distribution can then dynamically adapt based on usefulness.
- *PD-aware feedback.* Scalar fitness values are often insufficient to diagnose near-miss candidates. Placement-specific signals, such as displacement statistics and geometric pathologies, provide more actionable feedback for refinement.

Evolving OpenROAD's global and detailed placement modules under this loop yields implementations that improve both post-detailed-placement HPWL and routed wirelength relative to default OpenROAD across multiple design–node pairs (Figure 2).

Algorithm-level implication. LLM-driven code evolution can extend to compiled production EDA code when grounded in structured domain knowledge and closed with PD-aware feedback. The scope remains localized: the agent does not reason about cross-module interactions or repository-wide correctness obligations.

3.3 Repo-scale opt. as a codebase-level agent

Much of an EDA R&D engineer's work is inherently cross-module. A change in the resizer may interact with the placer's timing-weight logic, which can in turn affect downstream routing and final PPA. Reasoning across an entire production repository under QoR feedback, while preserving invariants that are not localized to any single file, is qualitatively harder than improving a single algorithm in isolation. This is the regime of codebase-level automation.

Agentic principles as scaffolding. Repository-scale optimization can be formulated as a structured coding workflow. An agent reads and indexes the repository, proposes research directions, expands them into implementation plans, localizes those plans to specific edit surfaces, and submits executable diffs under closed-loop QoR feedback. Three design choices make this workflow tractable:

- *Repository legibility.* The codebase representation must support planning and editing (e.g., graph-based representations of files, symbols, and dependencies, together with module-level documentation that summarizes relevant functionality and edit surfaces).
- *Structured planning.* The agent must emit explicit research plans before attempting edits. Each plan specifies objectives, hypotheses, target files, expected metrics, and validation criteria. This allows infeasible or low-value directions to be filtered out.
- *Formal gating.* Build success and QoR improvement are not sufficient acceptance criteria, since a QoR-improving patch may still introduce unsafe behavior. Candidate diffs must pass repository-level checks (e.g., compilation, tests, DRC-clean flow outcomes, etc.). For a subset of edits, the agent also emits a machine-checkable invariant, and the patch is accepted only if the associated proof obligation discharges in a pinned formal environment. This catches silent semantic failures that QoR gating alone may miss.

Under this regime, repository-scale agents discover reusable diffs that improve routed wirelength, effective clock period, and power across multiple OpenROAD modules, PDKs, and designs (Figure 3).

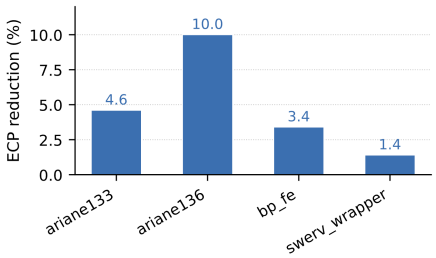

Figure 3: Effective-clock-period (ECP) reductions from repository-scale agentic optimization on four NG45 designs.

Codebase-level implication. Repository-scale modification under PD feedback is feasible when the codebase is made legible, planning is auditable, and acceptance combines QoR regression with correctness gates. This tier closely resembles EDA R&D engineering, but still lacks composition with design-task, algorithm-level, and flow-level agents.

4 Roadmap toward autonomous EDA R&D

Recent systems demonstrate feasibility at individual tiers, but they do not yet compose into a closed-loop automated EDA R&D engineer. Such a system would need to read a new idea, implement it across the relevant tool modules, integrate it into a flow, evaluate it on representative designs, and determine whether the idea produced a genuine improvement. Closing this loop requires six capabilities that no current system provides in unified form. These capabilities fall into three layers: evaluation mechanisms that produce reliable feedback; trust mechanisms that preserve correctness and provenance; and composition mechanisms that allow heterogeneous agents to operate in one shared loop.

- **Evaluation fidelity versus cost.** Full downstream P&R is too expensive for the inner loop of "long-horizon" R&D. Existing remedies, including proxy metrics, staged evaluation, and early ranking, are often ad hoc. A more principled approach requires an explicit *fidelity ladder* of increasingly faithful checks: linting, compilation, simulation, synthesis with lightweight estimates,

reduced-design physical checks, full-flow evaluation, and optional signoff-calibrated audit. Learned surrogates should be treated as first-class components of this environment, rather than as external predictors. The agent's decision problem then becomes not only how to improve QoR, but also which measurement provides the most useful information per unit cost.

- **Constraint-aware rewards.** Unlike SWE-bench-style settings, where task completion is largely binary, a PD flow may complete successfully while producing an unusable result. A candidate may violate timing, fail DRC/LVS, or regress power despite improving a target metric. Rewards should therefore be *lexicographic*: hard constraints such as syntax, simulation correctness, legality, DRC/LVS, etc. must precede normalized QoR improvement, which in turn must precede compute cost. Graders should report a feasibility envelope and a QoR frontier rather than a single scalar score. Otherwise, agents may learn superficial strategies (e.g., relaxing constraints or regressing unmeasured objectives).
- **Verification of algorithmic edits.** As Tier 2 agents grow in scope, build success and QoR regression are insufficient acceptance criteria. Formal contracts can capture narrow numerical invariants, but broader hazards remain (e.g., memory safety, undefined behavior, stale assumptions across modules). Trustworthy deployment will require a first-class verification substrate for agent-generated PD code, combining theorem proving, SMT-based checks, static analysis, and abstract interpretation.
- **Research-grade memory.** Current systems retain traces and artifacts within a run, but they do not accumulate knowledge across weeks or months in the way a human researcher does. An automated EDA R&D engineer needs persistent memory over hypotheses, failed directions, design-family effects, implementation decisions, and lessons extracted from prior experiments. Such memory must also be provenance-aware: the system should know which result supported which claim, under which tool version, design, PDK, and constraint set. Without this capability, agents will remain "short-horizon" optimizers.
- **Cross-tier orchestration.** No current system jointly coordinates Tier 3 design-task agents, Tier 2 algorithm-level agents, Tier 2 codebase-level agents, and Tier 1 flow-level agents in a shared loop. Yet this is how human EDA R&D proceeds: design insight, algorithmic change, repository integration, and flow-level evaluation inform one another. The missing component is an orchestrator with shared state, shared fitness signals, and mechanisms for arbitrating among conflicting proposals. This orchestration layer is the architectural gap between isolated per-tier automation and a robust automated researcher.
- **Machine-readable interfaces and episode schemas.** A closed-loop R&D agent needs stable and semantically meaningful access to the internal state of the flow, such as constraint provenance, incremental timing deltas, and placement pathologies. Today, much of this information is exposed only indirectly through logs, reports, and ad hoc scripts. The field needs a common *episode schema* for EDA R&D, together with structured APIs over reports, design states, tool actions, and evaluation traces. Vertical environments in adjacent domains (e.g., biomedical agent benchmarks [6]) suggest that curated tools and unified execution interfaces can potentially turn capable models into accountable agents.

The path to an automated EDA R&D engineer is therefore not simply a matter of scaling a single model. It is a systems problem: building the interfaces, rewards, memory, evaluation hierarchy, orchestration layer, and verification substrate that allow heterogeneous agents to participate in a common experimental loop. The next phase of agentic PD will likely be defined less by isolated capability demonstrations than by infrastructure for sustained, trustworthy, and cumulative research automation.

5 Conclusion

Agentic AI for EDA R&D has progressed far enough to make automation plausible across three core modes of practice: producing design artifacts, improving algorithms, and modifying production tool codebases. The central gap is the absence of a systems framework that composes these capabilities into a closed research loop. The remaining bottlenecks—evaluation efficiency, cross-tier orchestration, research-grade memory, and verification—are primarily systems-integration challenges rather than model-capability limitations. The next phase of agentic PD will therefore be defined less by larger base models than by infrastructure that makes agent capabilities accountable, composable, and cumulative.

References

[1] A. Agnesina, P. Rajvanshi, T. Yang, G. Pradipta, A. Jiao, B. Keller et al., "AutoDMP: automated DREAMPlace-based macro placement", *Proc. ISPD*, 2023.

[2] C.-K. Cheng, A. B. Kahng, I. Kang and L. Wang, "RePlAce: advancing solution quality and routability validation in global placement", *IEEE TCAD* 38(9) (2018), pp. 1717-1730.

[3] A. Ghose, J. Jang, A. B. Kahng and J. Lee, "Automated QoR improvement in OpenROAD with coding agents", *arXiv:2601.06268*, 2026, https://arxiv.org/abs/2601.06268.

[4] A. Ghose, A. B. Kahng, S. Kundu and B. Pramanik, "Invited: agentic AI for physical design R&D: status and prospects", *Proc. ISPD*, 2026, pp. 133–141.

[5] A. Ghose, A. B. Kahng, S. Kundu and Z. Wang, "ORFS-agent: tool-using agents for chip design optimization", *Proc. MLCAD*, 2025, pp. 1–13.

[6] K. Huang, S. Zhang, H. Wang, Y. Qu, Y. Lu, Y. Roohani et al., "Biomni: a general-purpose biomedical AI agent", *bioRxiv*, 2025.

[7] T. Jafri and V. A. Chhabria, "GR-Evolve: design-adaptive global routing via LLM-driven algorithm evolution", *arXiv:2604.22234*, 2026, https://arxiv.org/abs/2604.22234.

[8] C. E. Jimenez, J. Yang, A. Wettig, S. Yao, K. Pei, O. Press et al., "SWE-bench: can language models resolve real-world GitHub issues?", *arXiv:2310.06770*, 2024, https://arxiv.org/abs/2310.06770.

[9] A. B. Kahng, R. Varadarajan, and Z. Wang, "Hier-RTLMP: a hierarchical automatic macro placer for large-scale complex IP blocks", *IEEE Trans. on Computer-Aided Design of Integrated Circuits and Systems*, 43(5) (2024), pp. 1552–1565.

[10] A. Kaintura, Palaniappan R, S. S. Luar and I. Iyer Almeida, "ORAssistant: a custom RAG-based conversational assistant for OpenROAD", *arXiv:2410.03845*, 2024, https://arxiv.org/abs/2410.03845.

[11] F. Liu, X. Tong, M. Yuan, X. Lin, F. Luo, Z. Wang et al., "Evolution of heuristics: towards efficient automatic algorithm design using large language model", *arXiv:2401.02051*, 2024, https://arxiv.org/abs/2401.02051.

[12] M. Liu, N. Pinckney, B. Khailany and H. Ren, "ChipNemo: domain-adapted LLMs for chip design", *arXiv:2311.00176*, 2023, https://arxiv.org/abs/2311.00176.

[13] M. A. Merrill, A. G. Shaw, N. Carlini, B. Li, H. Raj, I. Bercovich et al., "Terminal-bench: benchmarking agents on hard, realistic tasks in command line interfaces", *arXiv:2601.11868*, 2026, https://arxiv.org/abs/2601.11868.

[14] A. Mirhoseini, A. Goldie, M. Yazgan, et al., "A graph placement methodology for fast chip design", *Nature* 594 (2021), pp. 207–212.

[15] A. Novikov, N. Vũ, M. Eisenberger, E. Dupont, P.-S. Huang, A. Z. Wagner et al., "AlphaEvolve: a coding agent for scientific and algorithmic discovery", *arXiv:2506.13131*, 2025, https://arxiv.org/abs/2506.13131.

[16] B. Romera-Paredes, M. Barekatain, A. Novikov, M. Balog, M. P. Kumar, E. Dupont et al., "Mathematical discoveries from program search with large language models", *Nature* 625(7995) (2024), pp. 468–475.

[17] 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 Trans. CAD* 43(9) (2024), pp. 2717–2730.

[18] B.-Y. Wu, A. Dey, A. Rovinski and V. A. Chhabria, "Focus session: large language models in physical design: from data generation to intelligent agents", *Proc. DATE*, 2026, pp. 1–7.

[19] S. Yao, J. Zhao, D. Yu, N. Du, I. Shafran, K. Narasimhan et al., "ReAct: synergizing reasoning and acting in language models", *arXiv:2210.03629*, 2024, https://arxiv.org/abs/2210.03629.

[20] X. Yao, J. Jiang, Y. Zhao, P. Liao, Y. Lin and B. Yu, "Evolution of optimization algorithms for global placement via large language models", *arXiv:2504.17801*, 2025, https://arxiv.org/abs/2504.17801.

[21] C. Yu and H. Ren, "Autonomous evolution of EDA tools: multi-agent self-evolved ABC", *arXiv:2604.15082*, 2026, https://arxiv.org/pdf/2604.15082.