PDAGENT-BENCH: CHARACTERIZING, GROUNDING, AND ARCHITECTING LLM/VLM AGENTS FOR VLSI PHYSICAL DESIGN

Anonymous authors
Paper under double-blind review

ABSTRACT

Large Language Models (LLMs) and vision-language models (VLMs) have shown remarkable success in the front-end design of Very Large-Scale Integrated Circuits (VLSI), yet their capabilities for VLSI physical design (PD) remain significantly underexplored. One primary cause is the lack of standardized benchmarks for evaluating agentic PD workflows that require high-dimensional, multi-stage optimization under strict design constraints, coordinated interaction with diverse Electronic Design Automation (EDA) tools, and iterative refinement. This work introduces **PDAGENT-BENCH**, a comprehensive and multi-dimensional **bench**mark for evaluating LLM/VLM-based **agent**s across the **p**hysical **d**esign stack. PDAGENT-BENCH integrates both task-level assessment and workflow-level execution. The benchmark suite contains 353 curated problems that combine conceptual questions with real-world industrial artifacts, with expert-validated references and executable solutions. **These tasks cover five key capability dimensions**: foundational knowledge, report comprehension, root-cause analysis, script generation, and full-flow implementation. In addition, the benchmark provides **a unified, human-aligned agentic PD workflow framework** that enables closed-loop evaluation of holistic PD in realistic EDA environments. Experiments on state-of-the-art models reveal that while modern LLMs/VLMs perform competitively on conceptual tasks (e.g., Opus5 achieves 74.8% on root-cause analysis), they remain substantially limited in tool-centric execution and long-horizon (Opus5 achieves 39.8% on Script-ICC2), multi-stage reasoning. Our studies further show that human-skill-enhanced agentic workflows significantly improve end-to-end PD performance. PDAGENT-BENCH establishes a standardized, reproducible, and realistic evaluation framework for advancing LLM/VLM-driven holistic physical design automation.

1 INTRODUCTION

Very Large-Scale Integrated Circuits (VLSI) are foundational hardware to power various modern technological advances, such as AI, scientific computing, data centers, and more (Weste & Harris, 2015; Kahng et al., 2011). The explosive growth of applications in these critical sectors is driving an unprecedented demand for VLSI design with higher productivity, better performance, and greater scale. Yet, conventional VLSI design workflows are increasingly unable to keep pace (Figure 1). A fundamental limitation lies in the **inherently iterative and human-in-the-loop design paradigm**, where engineers must repeatedly orchestrate, analyze, and refine design decisions across the entire stack, from front-end synthesis to back-end physical design (Yu, 2023). This tightly coupled, multi-stage workflow results in substantial engineering overhead, long feedback cycles, and limited scalability, particularly as design complexity continues to grow (Lavagno et al., 2018). Recent advances in large language models (LLMs) and vision-language models (VLMs) have opened a promising direction toward *agentic design automation*, in which LLM/VLM-based agents can iteratively reason, act, and optimize solutions in closed-loop settings. Specifically, in front-end design, LLMs have demonstrated strong capabilities in generating and debugging hardware code, significantly reducing manual effort (Ho et al., 2025; Zhu et al., 2025; Liu et al., 2023). For example, NVIDIA's ChipNeMo (Liu et al., 2023) shows that high-quality RTL (Register-Transfer Level) code can be generated with just a few prompts. Importantly, these advances have not emerged in

Reviewers: please read the Reviewer Guidelines (iclr.cc/Conferences/2027/ReviewerGuidelines) and the AI Policy for Reviewers (iclr.cc/Conferences/2027/AIPolicyForReviewers). If you used AI to expand, edit, or polish your review, please provide the input text to the LLM. Better still, consider skipping the LLM and submitting your original text: we, and the authors, are much more interested in your unedited thoughts than in what an LLM has to say. AI-assisted or not, you are putting your name and reputation behind your review: LLM-generated falsehoods, hallucinations or misrepresentations are subject to disciplinary action, which may include desk-rejecting all papers you have authored.

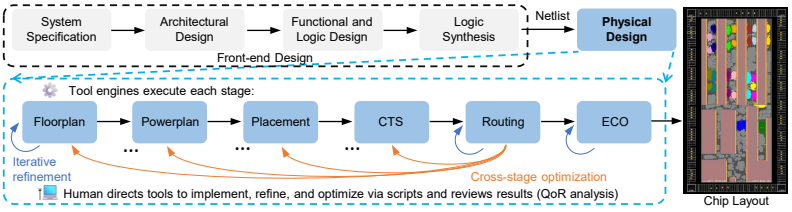


Figure 1: Modern VLSI design workflow. It spans from system specification through physical design, with physical design encompassing floorplan through engineering change order (ECO). Each stage involves iterative refinement within the stage (blue) and may trigger cross-stage optimization that loops back to earlier stages (orange). EDA tool engines execute each stage, while human engineers direct the tools via scripts and constraints, and review results with quality-of-results (QoR) analysis.

isolation; rather, they have been enabled and sustained by **well-defined benchmarks and evaluation frameworks** (Pinckney et al., 2024; Lu et al., 2023; Liu et al., 2025), which provide standardized tasks, reproducible protocols, and quantitative metrics for assessing progress.

Yet, **physical design remains largely underexplored with LLMs/VLMs**, despite its decisive role in determining a chip's final power, performance, and area (PPA). The primary barrier to progress is the **lack of standardized benchmarks** that can capture the full complexity of physical design and enable systematic evaluation of LLM/VLM-based agentic physical design workflows. It is important to note that physical design presents fundamentally different challenges from front-end tasks: **physical design involves complex geometric constraints, strict technology design rules, multi-tool orchestration, and long-latency feedback loops**. Thus, it requires integrating diverse cognitive capabilities: understanding design rules, interpreting tool-generated reports, diagnosing violations, generating tool scripts, and coordinating iterative optimization across stages. Existing benchmarks fail to capture this complexity. Prior works such as ChiPBench (Wang et al., 2024) focus on isolated subproblems (e.g., macro placement) with conventional algorithms, while ChatEDA-Bench (Wu et al., 2024) and iScript-Bench (Xu et al., 2026) evaluate narrow capabilities such as script generation. As a result, **current benchmarks are fragmented and disconnected from end-to-end workflows**, leaving them insufficient to assess whether LLM/VLM agents can effectively support realistic physical design.

In this work, we seek to evaluate the capabilities of LLM/VLM-based agents that are engineered to enable *holistic physical design* in realistic EDA environments. Formally, this pursuit requires assessing an agent's ability to (i) perform various types of tasks throughout the design stack, (ii) operate within multi-stage tool-driven workflows, and (iii) optimize design objectives (e.g., PPA) through iterative interaction with EDA tools. Achieving the goal requires a benchmark that satisfies several key desiderata. **(D1) Coverage.** It must span the full spectrum of physical design capabilities, beyond isolated tasks. **(D2) Workflow Fidelity.** Evaluation must reflect realistic multi-stage workflows with tool-in-the-loop feedback. **(D3) Standardization.** Tasks, inputs, and metrics must be reproducible and comparable across models. **(D4) Compositionality.** The benchmark should capture interactions between capabilities (e.g., report understanding $\rightarrow$ debugging $\rightarrow$ script generation $\rightarrow$ tool use). **(D5) Diagnostic Ability.** It should enable fine-grained analysis of failure modes and capability gaps.

To this end, we introduce **PDAGENT-BENCH**, a first-of-its-kind **benchmark for holistic and reproducible evaluation of LLM-based agentic physical design**. Our benchmark integrates both task-level evaluation and workflow-level execution. **Benchmark Design.** PDAGENT-BENCH consists of a **multi-dimensional benchmark suite** with 353 curated tasks spanning five capability dimensions: (i) foundational knowledge understanding, (ii) report comprehension, (iii) root-cause analysis, (iv) script generation, and (v) full-flow orchestration. Each task is designed to reflect realistic scenarios in physical design and is paired with standardized evaluation criteria. **Evaluation Framework.** We further develop a **unified human-aligned agentic design workflow framework** that enables closed-loop interaction between LLM/VLM agents and various EDA tools. This framework allows agents to iteratively **place macros**, generate actions (e.g., scripts), receive tool feedback (e.g., timing or congestion reports), and refine decisions, thereby enabling evaluation under realistic design loops rather than static prompts. Using this framework, we conduct a comprehensive evaluation on 16 state-of-the-art LLMs/VLMs. Our results provide the first systematic characterization of LLM/VLM

Table 1: Comparison of PDAGENT-BENCH with other circuit benchmarks.

Benchmark Features	VerilogEval V2 (Pinckney et al., 2024)	CircuitNet 3.0 (Wang et al.)	ChiPBench (Wang et al., 2024)	ChatEDA-Bench (Wu et al., 2024)	**PDAGENT-BENCH (Ours)**
Open Source	✓	✓	✓	✓	✓
RTL-to-GDS Flow	×	×	✓	✓	✓
Industrial Setting†	×	✓	×	×	✓
Domain Knowledge	×	×	×	×	✓
Report Analysis	×	×	×	×	✓
Root Cause Analysis	×	×	×	×	✓
Static Timing Analysis	×	×	×	×	✓
# of Data	153	8,659	20	50	353
Target Tasks	RTL Generation	Early-Stage Timing/Power Prediction	Macro Placement Optimization	Script Generation	**Comprehensive Physical Design (including macro placement)**

†**Industrial Setting**: uses commercial EDA tools on advanced technology nodes (≤ 28 nm).

capabilities in physical design, revealing critical limitations in long-horizon reasoning, tool interaction, and cross-stage coordination.

Our key contributions are summarized as follows: **(i)** We propose **PDAGENT-BENCH**, the first benchmark for **end-to-end agentic VLSI physical design workflows**, enabling standardized and reproducible evaluation. **(ii)** We construct a **353-task benchmark suite** covering five key capability dimensions, bridging isolated tasks and holistic workflows. **(iii)** We develop a **unified agent-in-the-loop framework** for closed-loop evaluation with EDA tools by plugging in existing LLMs/VLMs. **(iv) We present a systematic empirical study on 16 LLMs/VLMs, and identifying key capability gaps, failure modes, and critical insights for domain-specific model development**.

2 BACKGROUND AND RELATED WORK

VLSI Design Workflow. A typical VLSI design workflow used by human designers is shown in Figure 1, which has two phases: *front-end design* and *physical design*. The front-end phase includes architectural design, functional and logic design (hardware coding), and logic synthesis, where high-level functional specifications are translated into a gate-level netlist. This netlist serves as an input to the physical design phase, which generates a manufacturable layout. Physical design is a **multi-stage, tightly coupled optimization process** comprising several key steps (Lavagno et al., 2018). **Floorplanning** defines the spatial organization of macros and functional blocks, including core area, I/O pin locations, and macro placement, often requiring substantial manual guidance due to limited automation in current tools. **Power planning** constructs a hierarchical power delivery network (PDN) (e.g., rings, stripes, meshes, vias) to satisfy IR-drop and electromigration constraints while minimizing routing overhead. **Placement** assigns legal locations to standard cells, optimizing objectives such as wirelength, congestion, timing, and density. **Clock Tree Synthesis (CTS)** builds a buffered clock network to distribute the clock signal with controlled skew, latency, and transition characteristics. **Routing** establishes signal connectivity across metal layers under strict design-rule constraints, balancing wirelength, congestion, and manufacturability. **Engineering Change Order (ECO)** performs localized post-route modifications, including cell resizing, buffer insertion, and metal-only fixes, to resolve residual timing violations and design-rule errors without disrupting the global layout. Although each stage is supported by specialized EDA tools, achieving high-quality designs still relies heavily on **human experts to orchestrate the tools in the design workflow**. Importantly, these stages are **strongly interdependent**: early decisions (e.g., floorplanning) constrain downstream feasibility; placement impacts congestion and timing closure; and both influence the effectiveness of CTS, routing, and ECO. As a result, physical design requires **iterative, cross-stage refinement under long feedback loops**, making it a complex, feedback-driven process that is difficult to automate and scale.

Agentic Modeling and Benchmarking. LLM/VLM-based agents are well-suited for VLSI physical design due to two key properties in the physical design workflow. **(i) Script-driven, tool-mediated interaction.** Physical design stages are executed via parameterized tool commands (e.g., Tcl scripts), which encode key decisions such as PDN configuration, placement constraints, and CTS strategies. Consequently, the workflow can be abstracted as a sequence of *state-action-feedback* interactions, where an agent observes design states (e.g., reports, constraints), generates executable actions (scripts), and receives feedback from tools. This aligns closely with the LLM/VLM agent paradigm

of reasoning, tool invocation, and iterative refinement. **(ii) Structural regularity and compositional skill reuse.** Physical design exhibits strong regularity across designs within the same circuit class (e.g., CPUs, accelerators). Common patterns arise in floorplanning templates, PDN topologies, placement strategies, and timing closure heuristics. In practice, expert designers leverage this regularity by reusing and adapting prior solutions rather than solving each instance from scratch. This suggests that effective automation requires not only reasoning, but also **retrieval and composition of reusable skills** (e.g., script templates, optimization procedures). Such properties make physical design a natural setting for evaluating **generalization, compositionality, and skill reuse** in LLM agents. Together, these properties make physical design a natural setting for **closed-loop, tool-integrated evaluation** of LLM agents, motivating the need for a holistic benchmark for agentic physical design.

Related Work. Recently, several benchmarks (Lu et al., 2023; Liu et al., 2025; Pinckney et al., 2024; Wang et al.; 2024; Wu et al., 2024; Xu et al., 2026; Liu et al., 2019; Ghose et al., 2025a) have been proposed to evaluate LLM/VLM-based agents and conventional optimization algorithms for VLSI design (Table 1). VerilogEval v2 (Pinckney et al., 2024) is restricted to RTL code generation, while CircuitNet 3.0 (Wang et al.) is confined to early-stage timing and power prediction. ChipBench (Wang et al., 2024) focuses on macro-placement algorithms (analytical methods + reinforcement learning), and ChatEDA-Bench (Wu et al., 2024) and iScript-Bench (Xu et al., 2026) primarily evaluate EDA script generation. Despite these contributions, existing benchmarks share a key limitation: they evaluate **isolated capabilities** rather than **end-to-end workflows**. Beyond benchmarks, recent agent methods (Ghose et al., 2025b; Wu et al., 2025; Shrestha et al., 2026; 2024) such as ORFS-agent (Ghose et al., 2025b) and OpenROAD Agent (Wu et al., 2025) demonstrate closed-loop interaction with EDA tools, iteratively tuning flow parameters and generating self-correcting scripts from tool feedback, precisely the capabilities that existing benchmarks fail to capture. However, *none of the current physical design benchmarks capture the closed-loop interaction between agents and EDA tools, nor do they assess critical capabilities such as report comprehension, diagnostic reasoning, and iterative refinement across stages*. As a result, they fail to reflect the **feedback-driven, multi-stage nature** of real-world physical design, leaving a significant gap in the evaluation of whether LLM/VLM agents can operate effectively in practical settings. PDAGENT-BENCH addresses this critical gap by evaluating all these dimensions jointly (Table 1).

3 PDAGENT-BENCH

PDAGENT-BENCH consists of two complementary components: (i) a *multi-dimensional benchmark dataset* that captures the breadth and complexity of physical design tasks, and (ii) a *tool-integrated, multi-agent design workflow framework* that enables evaluation under realistic EDA interactions. Together, these components support **holistic, standardized, and reproducible evaluation** of LLM/VLM agents in holistic physical design.

3.1 BENCHMARK DEVELOPMENT

Overview. To systematically evaluate LLM/VLM capabilities in physical design, we construct a benchmark of 353 curated problems (Table 2). The dataset spans a spectrum from foundational knowledge questions to production-oriented tasks, including report comprehension, root-cause analysis, optimization reasoning, and script generation. This design aims to capture the core competencies required of a physical-design engineer while enabling fine-grained analysis of model performance.

Data Selection Criteria. Each problem in our benchmark dataset is selected to satisfy the following principles. **(C1) Coverage and diversity.** Tasks span multiple stages of the physical design flow and require both domain knowledge and structured reasoning. **(C2) Ground-truth fidelity.** Each instance is paired with a detailed reference solution to enable precise evaluation and error analysis. **(C3) Multi-modality.** Problems incorporate text, scripts, and visual elements (e.g., floorplans, design rule check (DRC) maps), reflecting real-world inputs. **(C4) Leakage resistance.** Instances are curated to minimize overlap with publicly available data, reducing the risk of memorization-based performance. **(C5) Realistic difficulty.** Tasks extend beyond simple question-answering (QA) to include complex operations such as timing report interpretation and full-flow reasoning.

Data Representation and Preprocessing. The raw dataset comprises 3 modalities, including scripts, free-form text, and images. To enable consistent processing and evaluation, we normalize all entries

Table 2: Overview of benchmark categories used to evaluate agents on physical design tasks. Report Comprehension, Root Cause Analysis, and Static Timing Analysis are multi-modal tasks that pair text with EDA reports, layouts, or timing diagrams. The targeted agent capabilities are denoted by: ① Reasoning, ② Planning, ③ Perception, ④ Tool Use.

Task	Category	#	Description	Agent Capability
Physical Design Understanding	Foundational Knowledge	90	Conceptual questions on physical-design and EDA fundamentals (timing, power, placement, routing, DRC/LVS).	①, ②
	Report Comprehension	11	Interpreting EDA tool reports (timing, congestion, power, utilization) to extract metrics and identify issues.	①, ③
	Root Cause Analysis	21	Diagnosing the underlying cause of PPA or convergence problems and proposing optimization directions.	①, ③
	Static Timing Analysis	11	Path-level reasoning over setup/hold violations, slack, clock skew, and timing exceptions.	①, ③
Script Generation	P&R Vendor 2	90	Generating Cadence Innovus TCL scripts for place-and-route flow steps.	④
	P&R Vendor 1	88	Generating Synopsys IC Compiler II TCL scripts for backend implementation.	④
	ECO	10	Generating engineering-change-order scripts for post-route fixes (timing, DRC, functional).	④
	FM (Formal Verification)	22	Generating Synopsys Formality scripts for RTL-vs-netlist equivalence checking.	④
Full Flow Implementation	End-to-End **placement** and **route (PnR)**	10	Driving a complete RTL-to-GDSII flow on the design suite below, evaluated on PPA, timing closure, and DRC/LVS cleanliness.	①, ②, ③, ④

Table 3: Benchmark designs used in the full-flow implementation. We report the cell count after synthesis, the target clock period, and the utilization target.

Design	Description	#Cells	Target Clk (ns)	Util.
TinyRISCV (Liang, 2020)	Compact RISC-V core (small control-logic design).	15,012	5.0	0.65
Open910 (Chen et al., 2020)	Open-source application-class processor.	754,981	1.5	0.60
Systolic array (Abdelazeem, 2021)	Matrix-multiply accelerator (datapath-heavy).	41,198	3.4	0.50
16b pipelined mult. (OpenCores, 2001)	16-bit pipelined multiplier (arithmetic datapath).	342	2.6	0.55
AES-256 (Kindér, 2014)	AES-256 cryptographic core.	15,516	3.5	0.60
NVDLA-small (NVIDIA, 2017)	NVIDIA Deep Learning Accelerator, small config.	270,072	2.0	0.60
NVDLA-large (NVIDIA, 2017)	NVIDIA Deep Learning Accelerator, large config.	1,478,865	2.5	0.60
UART controller (Ultra-Embedded, 2020)	Serial communication peripheral.	368	3.4	0.65
Elevator FSM (OpenCores, 2001)	Finite-state-machine control design.	176	2.5	0.70
Ethernet MAC (OpenCores, 2001)	Ethernet media access controller.	35,172	10.0	0.60

into a **unified JSON schema**, ensuring structured representation and efficient parsing. For **script generation tasks**, we provide a comprehensive set of acceptable solutions, accounting for multiple valid command variants due to tool-version differences, alternative syntactic forms, and synonymous expressions. For **question-answering tasks**, we design detailed evaluation rubrics with clearly defined scoring criteria and key points, enabling reliable and consistent **LLM-as-judge** assessment.

Benchmark. Following the criteria and preprocessing strategies, we construct PDAGENT-BENCH benchmark dataset, summarized in Table 2. The benchmark comprises two complementary components. **First, foundational suite (90 instances)**. We curate 90 foundational questions that cover the full physical design flow, from initialization to tape-out, ensuring broad coverage of core concepts and design stages. **Second, production-oriented suite (263 instances)**. To evaluate performance on real-world, production-oriented tasks, we include 263 additional instances derived from commonly used implementation scripts, timing reports, and error/violation reports collected from previous successful projects, whose results have been published at top-tier solid-state circuit design conferences including ISSCC (Anonymous, 2026), VLSI Symposium (Anonymous, 2025b), A-SSCC (Anonymous, 2024), and CICC (Anonymous, 2025a). The script generation spans the major commercial EDA tools used across the modern physical-design flow. To ensure quality and reliability, all questions and reference answers are independently reviewed by three domain experts. In addition, all script-based solutions are validated through execution, confirming functional correctness and practical usability within a simulation environment. We organize the benchmark into three evaluation suites. The **Physical Design Understanding suite (133 instances)** evaluates foundational concepts, EDA report comprehension, root-cause analysis, and static timing analysis. The **Script Generation suite (210 instances)** tests executable Tcl generation across the major commercial EDA tools. The **Full-Flow**

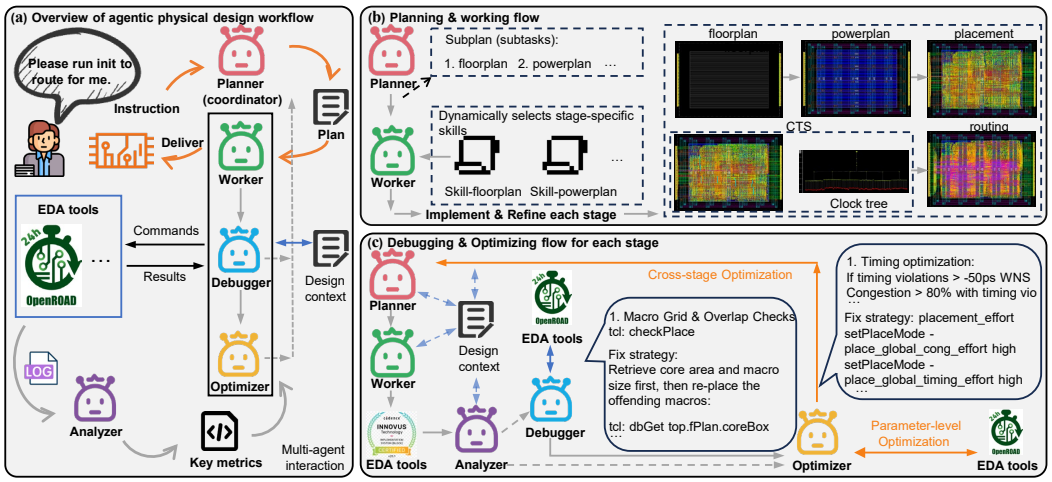


Figure 2: (a) Overview of the PDAgent framework. Five specialized agents (Planner, Worker, Debugger, Optimizer, and Analyzer) coordinate to translate user instructions into a complete physical-design flow. (b) The Planner decomposes the task into stage-wise subplans, and the Worker invokes stage-specific skills for each stage, advancing only after debugging and optimization are complete. (c) The Optimizer and Debugger refine the design through targeted fix strategies until convergence.

Implementation suite (Table 3) drives a complete RTL-to-GDSII (physical layout) flow on 10 designs, ranging from a 176-cell FSM (finite state machine) to a 1.5M-cell NVDLA accelerator, graded on placement quality, timing closure, DRC (design rule check)/LVS (layout vs. schematic) cleanliness, and PPA. As summarized in Table 2, **our benchmark is designed to probe the key capabilities required of agents, spanning four core aspects: planning, reasoning, tool use, and perception** (Zhang et al., 2025).

3.2 Agentic Design Workflow Framework

Overview. To enable systematic benchmarking of human-aligned agentic workflows for VLSI physical design, we develop **PDAgent**, a unified framework for constructing and evaluating LLM/VLM-based agents in closed-loop, tool-integrated environments (Figure 2(a)). The framework defines standardized **agent roles, interfaces, and coordination mechanisms**, allowing researchers to instantiate and compare diverse agentic workflows under consistent conditions. It is designed to be **modular and extensible**, supporting plug-and-play integration of different models (e.g., open-source or proprietary LLMs/VLMs) and customization for varying design objectives.

Given a gate-level netlist, technology constraints, and design targets, PDAgent orchestrates agents to iteratively produce a DRC-clean, timing-closed layout. The workflow follows the natural structure of physical design, comprising four phases, *planning, implementation, debugging, and optimization*, which form a closed loop for progressive refinement. Specifically, PDAgent comprises a Planner Agent, Worker Agent, Debugger Agent, and Optimizer Agent, augmented by an Analyzer Agent that extracts key metrics from verbose EDA tool logs. We briefly introduce the key design knobs of PDAgent below. Detailed descriptions of each agent's role are provided in Appendix A.9.

Agent Roles. The PDAgent comprises five agents, each encapsulating a distinct concern: **(i)** *Planner* serves as the central hub, decomposing user intent into a directed acyclic graph (DAG) of subtasks, routing all inter-agent messages, and maintaining global execution; **(ii)** *Worker* interfaces with the EDA tool (e.g., Innovus) via a persistent interactive shell, executing TCL commands and returning structured results, illustrated in Figure 2(b). Notably, **macro placement during the floorplan stage is performed directly by the NLMs agents**, which takes the layout image as visual input to guide placement decisions (see Appendix A.9.1 for details); **(iii)** *Analyzer* extracts quantitative metrics, timing, for example, worst negative slack (WNS), physical (DRC, utilization), and power, from EDA logs; **(iv)** *Debugger* diagnoses failures through LLM/VLM-powered root-cause analysis;

(v) *Optimizer* performs parameter tuning guided by metric trajectories and domain knowledge, as shown in Figure 2(c).

Hub-and-Spoke Communication. All inter-agent messages are routed through the Planning Agent, which maintains full observability of the system state and enforces execution ordering. The sole exception is *tool interaction*: any agent may directly invoke EDA tool commands via a synchronous `execute_tool` interface, bypassing hub routing to avoid latency overhead for frequent tool queries. This hybrid topology balances centralized coordination with efficient tool interaction.

Skill Packs. Building on the inherent skill-driven physical design workflow introduced in Section 2, the agentic framework is *grounded* in a curated skill library that encodes the procedural knowledge of senior engineers, yielding a skill-enhanced LLM/VLM-based agentic framework. Skills encode procedural knowledge, executable scripts together with the rationale behind parameter choices and PPA optimization strategies, distilled from senior engineers. This kind of tacit, experience-driven expertise is what traditional algorithmic approaches cannot capture, and what motivates an LLM/VLM-based agentic framework. Domain knowledge is organized into *skill packs*, self-contained directories structured. Each skill pack contains per-stage skill files that encode expert knowledge: reference TCL scripts, parameter-tuning strategies, debug decision matrices, and cross-stage impact analyses. Skills are auto-discovered at initialization and resolved dynamically via fuzzy matching, enabling seamless extension to new technology nodes and EDA tools without modifying agent code.

Closed-Loop PPA Optimization. Aligning with the practical VLSI design, the optimization mechanism operates at two levels: a **stage-level** loop that progresses through the physical design flow step by step, and a **parameter-level** loop that explores the design space within each stage. When design targets are unmet, the agent employs a **tree-of-thought exploration strategy**, generating and evaluating multiple candidate actions before selecting the most promising trajectory. Iteration continues until convergence or a predefined budget is reached.

4 Experiments and Results

This section presents comprehensive evaluations of various LLM/VLM-based agents on VLSI physical design tasks by using PDAgent-Bench. We first describe the experimental setup, then report the main results, and finally summarize key findings and insights.

4.1 Experimental Setup

Models and Computing Platforms. We evaluate 16 LLMs/VLMs with PDAgent-Bench, including 10 proprietary and 6 open-source models. The proprietary models are GPT-5-mini, GPT-5.1, GPT-5.4-mini, GPT-5.5 (Singh et al., 2025), GPT-5.6 Sol, GPT-5.6 Luna, Claude Sonnet 4.6 (Anthropic, 2026b), Claude Opus 4.7 (Anthropic, 2026a), Claude Sonnet 5, and Claude Opus 5. The open-source models are Qwen3-VL-8B, Qwen3-VL-30B (Bai et al., 2025), Qwen3-8B (Yang et al., 2025), DeepSeek-Chat, DeepSeek-Reasoner (Liu et al., 2024), and DeepSeek-V4.1-Flash. The five most recent models (GPT-5.6 Sol, GPT-5.6 Luna, Claude Sonnet 5, Claude Opus 5, and DeepSeek-V4.1-Flash) are evaluated in a separate round through their official APIs as single-turn queries without tools or retrieval, using medium reasoning effort and a 32K-token output cap. All experiments are done with 12 NVIDIA A6000 Ada GPUs and an AMD EPYC 7313 CPU.

EDA Tools and Semiconductor Nodes. We use Cadence Innovus 22 (Cadence), Synopsys IC Compiler II (ICC2) 2022 (Synopsys, b), Synopsys PrimeTime (PT) 2022 (Synopsys, c), and Synopsys Formal (FM) 2022 (Synopsys, a) as our commercial EDA tools for implementation and signoff. Command-level differences between tool versions are minimal; we have validated that our generated scripts are compatible with versions 2018 and 2020. We target the TSMC 28-nm process.

Evaluation Metrics and Sampling Configuration. We report **pass@1** and **pass@5** for *Physical Design Understanding* and *Script Generation* tasks. For free-response foundational knowledge, root-cause analysis, and report comprehension tasks, we generate five responses per query and score each against the reference answer using predefined rubrics. Pass@1 is obtained from a single response generated with temperature 0 (deterministic decoding). Pass@5 is computed by generating five responses with temperature 0.8 (if available) (top-p = 0.95) and reporting the average rubric score. The five recent models accept only their default sampling temperature, so both their pass@1

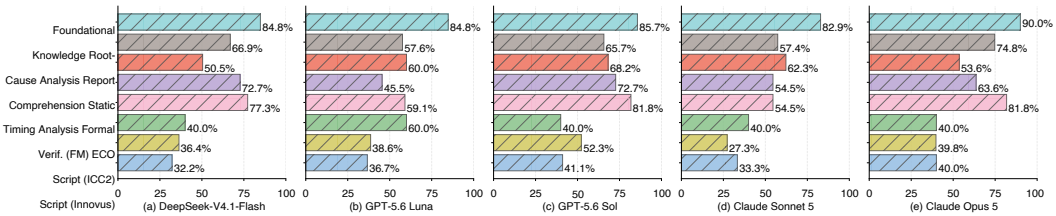


Figure 3: Per-task pass@1 (%) of five recent frontier LLMs across eight evaluation categories. Scores are high on foundational knowledge, but no model exceeds 53% on script generation or 60% on ECO. Full results are in Table 4.

Table 4: Benchmark results of five recent frontier LLMs across Physical Design Understanding and Script Generation. Each cell reports pass@1 / pass@5. Foundational, RootOpt, and Report are scored 0–10 by a blind GPT-5.4-mini rubric judge (shown divided by 10; pass@5 is the mean over five samples); the remaining tasks are graded by rule-based checkers (pass@5: at least one of five samples correct). All models are queried through their APIs with medium reasoning effort, a 32K-token output cap, and no tools. **Bold** indicates the best result per column; underline indicates the second best. Results for all earlier models are in Table 6 (Appendix A.4).

Model	**Foundational**		**RootOpt**		**Report**		**STA**		**FM**		**ECO**		**ICC2**		**Innovus**	
	p@1	p@5	p@1	p@5	p@1	p@5	p@1	p@5	p@1	p@5	p@1	p@5	p@1	p@5	p@1	p@5
DeepSeek-V4.1-Flash	.848	.863	.669	.659	.505	.566	**.727**	**.727**	.773	**.955**	.400	.500	.364	.466	.322	.444
GPT-5.6 Luna	.848	.829	.576	.590	.600	.604	.455	.545	.591	.773	**.600**	.500	.386	.455	.367	.467
GPT-5.6 Sol	.857	.871	.657	.671	**.682**	.658	**.727**	.636	**.818**	.818	.400	.400	**.523**	**.602**	**.411**	**.511**
Claude Sonnet 5	.829	.832	.574	.579	.623	.608	.545	**.727**	.545	.636	.400	**.600**	.273	.341	.333	.367
Claude Opus 5	**.900**	**.908**	**.748**	**.785**	.536	**.684**	.636	**.727**	**.818**	.909	.400	.400	.398	.534	.400	.444

and pass@5 responses are sampled at that default. For full-flow implementation tasks, we report **stage-specific design metrics**, including WNS, placement density, and the number of DRC violations, which directly reflect design quality and PPA outcomes.

LLM-as-Judge and Human Evaluation. To enable affordable, large-scale qualitative analysis of model outputs, we adopt the widely used LLM-as-Judge protocol (Zheng et al., 2023) with Claude Sonnet 4.6 as the judge. For the five recent models, we use GPT-5.4-mini as the judge with the same rubrics; the judge is blind to model identity, as self-identifying text is removed and responses from all models are shuffled before judging. Full prompts are provided in Appendix A.10. To validate our LLM-Judge protocol, we recruit three senior physical-design engineers to independently score the model outputs on Basic, RootOpt, and Report using the same rubric provided to the LLM judge. We report the mean score across the three engineers as the human-evaluation result in Table 6.

4.2 Experimental Results

Physical Design Understanding and Script Generation. Table 4 and Figure 3 report the results of the five most recent models. Claude Opus 5 leads on physical design understanding, with 90.0% pass@1 on foundational knowledge and 74.8% on root-cause analysis, while GPT-5.6 Sol leads on report comprehension (68.2%) and script generation (52.3% on ICC2 and 41.1% on Innovus). The open-source DeepSeek-V4.1-Flash is competitive with the proprietary models, ranking second on root-cause analysis (66.9%) and tying for the best STA score (72.7%). On the rule-graded tasks, which are scored identically in both evaluation rounds (Table 6, Appendix A.4), the newer models improve only modestly over their predecessors: the best ICC2 pass@1 rises from 45.5% (GPT-5.5) to 52.3%, the best Innovus pass@1 does not improve (42.2% vs. 41.1%), and the best STA and FM scores are unchanged (72.7% and 81.8%). Even with five samples, the best model solves only 60.2% of ICC2 and 51.1% of Innovus tasks, indicating that *tool-specific script generation remains the main bottleneck*.

Full-Flow Implementation. Table 5 reports partial full-flow implementation results of PDAgent on three representative designs from the PDAgent-Bench full-flow suite (i.e., TinyRISCV, AES-256, and

Table 5: Partial results of **PDAgent** on the PDAgent-Bench Full flow suite (TSMC 28nm); the complete results are reported in Appendix A.6. WNS and DRC violations are reported at each stage; positive WNS indicates timing closure. Utilization (Util.) is reported after floorplanning.

Design	Init	Floorplan	Powerplan	Place	CTS	CTSopt	Route	Routeopt
TinyRISCV	5ns, WNS: 1.362ns Setup successfully	Util: 0.65 DRC Vio: 0	DRC Vio: 0	WNS : -0.014ns DRC Vio: 0 Density: 0.715	WNS : -0.127ns DRC Vio: 0 Density: 0.7165	WNS : 0.001ns DRC Vio: 0 Density: 0.713	WNS : 0.021ns DRC Vio: 1 Density: 0.722	WNS : 0.021ns DRC Vio: 0 Density: 0.722
AES-256	3.5ns, WNS: 1.003ns Setup successfully	Util: 0.60 DRC Vio: 0	DRC Vio: 0	WNS : 0.003ns DRC Vio: 0 Density: 0.613	WNS : 0.006ns DRC Vio: 0 Density: 0.615	WNS : 0.006ns DRC Vio: 0 Density: 0.615	WNS : 0.12ns DRC Vio: 303 Density: 0.615	WNS : 0.119ns DRC Vio: 0 Density: 0.615
Ethernet MAC	10ns, WNS: 5.621ns Setup successfully	Util: 0.60 DRC Vio: 0	DRC Vio: 0	WNS : 5.62ns DRC Vio: 25 Density: 0.62	WNS : 5.475ns DRC Vio: 23 Density: 0.624	WNS : 5.470ns DRC Vio: 0 Density: 0.628	WNS : 5.53ns DRC Vio: 69 Density: 0.624	WNS : 5.53ns DRC Vio: 0 Density: 0.625

Ethernet MAC); complete results and an ablation against an LLM/VLM agent without our skill library are provided in the Appendix A.6. The skill-free baseline fails at early stages such as initialization and floorplanning, consistent with the weak script-generation performance reported in Table 4. In contrast, PDAGENT closes timing (positive WNS) and resolves all DRC violations after RouteOpt across the three designs, demonstrating that our agent can drive a complete RTL-to-GDSII flow on real industrial designs in TSMC 28 nm.

4.3 KEY FINDINGS AND INSIGHTS

Key Findings. We analyze the quantitative results alongside error cases (Appendix A.5) and summarize the findings, revealing the main challenges and directions for improving LLM-based agentic workflows in physical design. **Finding 1: Failures are predominantly syntactic rather than conceptual.** Approximately 92% of script-generation failures are near-misses, where models select the correct command (e.g., in Cadence Innovus) but produce incorrect, missing, or inconsistent arguments. This suggests that LLMs largely capture the *procedural structure* of physical design workflows, but lack fidelity in tool-specific syntax. Bridging this gap requires explicit **syntax grounding** or retrieval over tool documentation. **Finding 2: Strong performance on foundational knowledge.** On the foundational benchmark, GPT-5.5 achieves 80.4% overall across 90 questions, including 83.0% on easy and 81.7% on medium difficulty. **Finding 3: Perception outpaces decision-making.** While models achieve near-perfect performance on report comprehension ($\sim$9/10), performance drops significantly ($\sim$5/10) when tasks require root-cause diagnosis or optimization actions. **Finding 4: Skill augmentation enables end-to-end execution.** Integrating curated skill libraries substantially improves performance on full-flow tasks by reducing reliance on fragile script generation.

Insights. These findings suggest two complementary directions for advancing LLM/VLM-based physical design. *(i) Data and training.* Expanding datasets that align reports with root-cause analyses, optimization rationales, and executable scripts can directly address gaps in syntax fidelity and decision-making (**Findings 1 and 3**). *(ii) Skill-centric architectures.* Developing richer and more scalable skill libraries, along with efficient retrieval and composition mechanisms, can improve generalization across designs, tools, and technology nodes (**Finding 4**). Finally, training domain-adapted models on such curated data would enable **on-premises deployment**, which is essential for industrial physical design workflows involving proprietary designs and process design kits (PDKs). Limitations are discussed in Appendix A.2.

5 CONCLUSION

In this work, we introduce PDAGENT-BENCH, the first comprehensive benchmark for systematic evaluation of LLM/VLMs agents in VLSI physical design, comprising 353 tasks across five capability axes and a unified multi-agent framework for closed-loop interaction with EDA tools. Our evaluation of 16 frontier LLMs/VLMs reveals a substantial gap between conceptual understanding and tool-grounded execution. We anticipate PDAGENT-BENCH will accelerate the development of domain-specific LLM/VLMs agents for this underexplored but high-impact domain.

REFERENCES

Ahmed Abdelazeem. Systolic array implementation in RTL for TPU. https://github.com/abdelazeem201/Systolic-array-implementation-in-RTL-for-TPU, 2021.

Accessed: Apr. 2026.

Anonymous. Anonymized for double-blind review, 2024. IEEE Asian Solid-State Circuits Conference (A-SSCC).

Anonymous. Anonymized for double-blind review, 2025a. IEEE Custom Integrated Circuits Conference (CICC).

Anonymous. Anonymized for double-blind review, 2025b. Symposium on VLSI Technology and Circuits.

Anonymous. Anonymized for double-blind review, 2026. IEEE International Solid-State Circuits Conference (ISSCC).

Anthropic. Introducing claude opus 4.7, 2026a. URL https://www.anthropic.com/news/claude-opus-4-7.

Anthropic. Introducing claude sonnet 4.6, 2026b. URL https://www.anthropic.com/news/claude-sonnet-4-6.

Shuai Bai, Yuxuan Cai, Ruizhe Chen, Keqin Chen, Xionghui Chen, Zesen Cheng, Lianghao Deng, Wei Ding, Chang Gao, Chunjiang Ge, et al. Qwen3-vl technical report. *arXiv preprint arXiv:2511.21631*, 2025.

Cadence. Innovus implementation system. URL https://www.cadence.com/en_US/home/tools/digital-design-and-signoff/soc-implementation-and-floorplanning/innovus-implementation-system.html.

Chen Chen, Xiaoyan Xiang, Chang Liu, Yunhai Shang, Ren Guo, Dongqi Liu, Yimin Lu, Ziyi Hao, Jiahui Luo, Zhijian Chen, et al. Xuantie-910: A commercial multi-core 12-stage pipeline out-of-order 64-bit high performance risc-v processor with vector extension: Industrial product. In *2020 ACM/IEEE 47th Annual International Symposium on Computer Architecture (ISCA)*, pp. 52–64. IEEE, 2020.

Amur Ghose, Andrew B. Kahng, Sayak Kundu, Yiting Liu, Bodhisatta Pramanik, Zhiang Wang, and Dooseok Yoon. Use cases and deployment of ml in ic physical design. In *Proceedings of the 30th Asia and South Pacific Design Automation Conference*, ASPDAC '25, pp. 669–675, New York, NY, USA, 2025a. Association for Computing Machinery. ISBN 9798400706356. doi: 10.1145/3658617.3703143. URL https://doi.org/10.1145/3658617.3703143.

Amur Ghose, Andrew B. Kahng, Sayak Kundu, and Zhiang Wang. Orfs-agent: Tool-using agents for chip design optimization. In *2025 ACM/IEEE 7th Symposium on Machine Learning for CAD (MLCAD)*, pp. 1–13, 2025b. doi: 10.1109/MLCAD65511.2025.11189204.

Chia-Tung Ho, Haoxing Ren, and Brucek Khailany. Verilogcoder: Autonomous verilog coding agents with graph-based planning and abstract syntax tree (ast)-based waveform tracing tool. In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 39, pp. 300–307, 2025.

Andrew B Kahng, Jens Lienig, Igor L Markov, and Jin Hu. *VLSI physical design: from graph partitioning to timing closure*, volume 312. Springer, 2011.

Joachim Kindér. AES: Verilog implementation of the advanced encryption standard (AES-256). https://github.com/secworks/aes, 2014. Accessed: Apr. 2026.

Luciano Lavagno, Louis Scheffer, and Grant Martin. *EDA for IC implementation, circuit design, and process technology*. CRC press, 2018.

Kangnan Liang. TinyRISC-V: A simple RISC-V core. https://github.com/liangkangnan/tinyriscv, 2020. Accessed: Apr. 2026.

Aixin Liu, Bei Feng, Bing Xue, Bingxuan Wang, Bochao Wu, Chengda Lu, Chenggang Zhao, Chengqi Deng, Chenyu Zhang, Chong Ruan, et al. Deepseek-v3 technical report. *arXiv preprint arXiv:2412.19437*, 2024.

Mingjie Liu, Teodor-Dumitru Ene, Robert Kirby, Chris Cheng, Nathaniel Pinckney, Rongjian Liang, Jonah Alben, Himyanshu Anand, Sanmitra Banerjee, Ismet Bayraktaroglu, et al. Chipnemo: Domain-adapted llms for chip design. *arXiv preprint arXiv:2311.00176*, 2023.

Shang Liu, Yao Lu, Wenji Fang, Mengming Li, and Zhiyao Xie. Openllm-rtl: Open dataset and benchmark for llm-aided design rtl generation, 2025. URL https://arxiv.org/abs/2503.15112.

Wen-Hao Liu, Stefanus Mantik, Wing-Kai Chow, Yixiao Ding, Amin Farshidi, and Gracieli Posser. Ispd 2019 initial detailed routing contest and benchmark with advanced routing rules. In *Proceedings of the 2019 international symposium on physical design*, pp. 147–151, 2019.

Yao Lu, Shang Liu, Qijun Zhang, and Zhiyao Xie. Rtllm: An open-source benchmark for design rtl generation with large language model, 2023. URL https://arxiv.org/abs/2308.05345.

NVIDIA. NVDLA: Open source deep learning accelerator. https://github.com/nvdla/hw, 2017. Accessed: Apr. 2026.

OpenCores. Ethernet MAC 10/100 Mbps. https://opencores.org/projects/, 2001. Accessed: Apr. 2026.

Nathaniel Pinckney, Christopher Batten, Mingjie Liu, Haoxing Ren, and Brucek Khailany. Revisiting verilogeval: Newer llms, in-context learning, and specification-to-rtl tasks. *arXiv preprint arXiv:2408.11053*, 2024.

Pratik Shrestha, Alec Aversa, Saran Phatharodom, and Ioannis Savidis. Eda-schema: A graph datamodel schema and open dataset for digital design automation. In *Proceedings of the Great Lakes Symposium on VLSI 2024*, GLSVLSI '24, pp. 69–77, New York, NY, USA, 2024. Association for Computing Machinery. ISBN 9798400706059. doi: 10.1145/3649476.3658718. URL https://doi.org/10.1145/3649476.3658718.

Pratik Shrestha, Alec Aversa, and Ioannis Savidis. Eda-schema-v2: A multimodal schema, open datasets, and benchmarks for machine learning in digital physical design, 2026. URL https://arxiv.org/abs/2605.06952.

Aaditya Singh, Adam Fry, Adam Perelman, Adam Tart, Adi Ganesh, Ahmed El-Kishky, Aidan McLaughlin, Aiden Low, AJ Ostrow, Akhila Ananthram, et al. Openai gpt-5 system card. *arXiv preprint arXiv:2601.03267*, 2025.

Synopsys. Vc formal: Formal verification solution, a. URL https://www.synopsys.com/verification/static-and-formal-verification/vc-formal.html.

Synopsys. Ic compiler ii: Place & route solution, b. URL https://www.synopsys.com/implementation-and-signoff/physical-implementation/ic-compiler.html.

Synopsys. Primetime static timing analysis, c. URL https://www.synopsys.com/implementation-and-signoff/signoff/primetime.html.

Ultra-Embedded. UART controller IP core. https://github.com/ultraembedded/cores, 2020. Accessed: Apr. 2026.

Mingjun Wang, Yihan Wen, Yuntao Lu, Fengrui Liu, Yuxiang Zhao, Boyu Han, Jianan Mu, Yibo Lin, Runsheng Wang, Bei Yu, et al. Circuitnet 3.0: A multi-modal dataset with task-oriented augmentation for ai-driven circuit design. In *The Fourteenth International Conference on Learning Representations*.

Zhihai Wang, Zijie Geng, Zhaojie Tu, Jie Wang, Yuxi Qian, Zhexuan Xu, Ziyan Liu, Siyuan Xu, Zhentao Tang, Shixiong Kai, et al. Benchmarking end-to-end performance of ai-based chip placement algorithms. *arXiv preprint arXiv:2407.15026*, 2024.

Neil HE Weste and David Harris. *CMOS VLSI design: a circuits and systems perspective*. Pearson Education India, 2015.

Cognitive Silicon Design Workshop. Toward cognitive silicon design: Structural barriers and an industry roadmap for ai-native chip design. URL chrome-extension://efaidnbmnnnibpcajpcglclefindmkaj/https://vlsicad.ucsd.edu/NEWS26_2/TowardsCognitiveSiliconDesign-04032026_Workshop_Final.pdf.

Bing-Yue Wu, Utsav Sharma, Austin Rovinski, and Vidya A. Chhabria. Openroad agent: An intelligent self-correcting script generator for openroad. In *2025 IEEE International Conference on LLM-Aided Design (ICLAD)*, pp. 16–22, 2025. doi: 10.1109/ICLAD65226.2025.00039.

Haoyuan Wu, Zhuolun He, Xinyun Zhang, Xufeng Yao, Su Zheng, Haisheng Zheng, and Bei Yu. Chateda: A large language model powered autonomous agent for eda. *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, 43(10):3184–3197, 2024.

Ning Xu, Zhaoyang Zhang, Senlin Shu, Lei Qi, Jiaqi Lv, Wensuo Wang, Tianhao Zhao, Chao Zhang, Zhaoliang Yang, Xiangyu Li, et al. iscript: A domain-adapted large language model and benchmark for physical design tcl script generation. *arXiv preprint arXiv:2603.04476*, 2026.

An Yang, Anfeng Li, Baosong Yang, Beichen Zhang, Binyuan Hui, Bo Zheng, Bowen Yu, Chang Gao, Chengen Huang, Chenxu Lv, et al. Qwen3 technical report. *arXiv preprint arXiv:2505.09388*, 2025.

Bei Yu. Machine learning in eda: When and how. In *2023 ACM/IEEE 5th Workshop on Machine Learning for CAD (MLCAD)*, pp. 1–6. IEEE, 2023.

Guibin Zhang, Hejia Geng, Xiaohang Yu, Zhenfei Yin, Zaibin Zhang, Zelin Tan, Heng Zhou, Zhongzhi Li, Xiangyuan Xue, Yijiang Li, et al. The landscape of agentic reinforcement learning for llms: A survey. *arXiv preprint arXiv:2509.02547*, 2025.

Lianmin Zheng, Wei-Lin Chiang, Ying Sheng, Siyuan Zhuang, Zhanghao Wu, Yonghao Zhuang, Zi Lin, Zhuohan Li, Dacheng Li, Eric Xing, et al. Judging llm-as-a-judge with mt-bench and chatbot arena. *Advances in neural information processing systems*, 36:46595–46623, 2023.

Yaoyu Zhu, Di Huang, Hanqi Lyu, Xiaoyun Zhang, Chongxiao Li, Wenxuan Shi, Yutong Wu, Jianan Mu, Jinghua Wang, Yang Zhao, et al. Qimeng-codev-r1: Reasoning-enhanced verilog generation. *arXiv preprint arXiv:2505.24183*, 2025.

A APPENDIX

A.1 STATEMENT ON THE USE OF LARGE LANGUAGE MODELS

We used large language models (LLMs) for two purposes. First, for writing assistance, that is, improving the grammar, wording, and clarity of the manuscript. Second, for scoring: LLMs serve as automated judges that grade model responses on the free-response tasks (Foundational, RootOpt, and Report) using predefined rubrics, following the LLM-as-Judge protocol. The LLMs and VLMs evaluated on PDAGENT-BENCH, and those used as the backbone of PDAGENT, are the subjects of this study rather than research tools. The research idea, benchmark construction, experiments, and analysis were developed by the authors, who take full responsibility for the content of this paper.

A.2 LIMITATIONS AND FUTURE WORK

Limitations. While PDAGENT-BENCH provides a comprehensive framework for evaluating LLM/VLMs agents in physical design, several limitations remain. First, **signoff completeness** is not fully covered. Layout-versus-Schematic (LVS) verification is currently excluded, leaving a critical stage of the design flow unmodeled. Second, **automation coverage** is incomplete for certain tasks, particularly DRC fixing that relies on GUI-based interactions or manual inspection, which are difficult to capture within script-driven workflows. Third, **dataset scale and diversity** are limited: although the benchmark includes both foundational and production-oriented tasks, it does not yet fully reflect the scale, heterogeneity, and corner cases of industrial-grade designs.

Future Work. We outline several directions to extend both the benchmark and the agentic design framework. **First**, we plan to expand **technology coverage** beyond 28-nm to include both advanced nodes (e.g., 16-nm, 7-nm) and mature nodes (e.g., 65-nm, 130-nm), enabling broader evaluation across design regimes. **Second**, we will increase **dataset scale and complexity** by incorporating larger and more diverse designs, improving the benchmark's ability to stress-test long-horizon reasoning and workflow robustness. **Third**, we aim to develop more **fine-grained, stage-specific capabilities**, including automated macro placement, congestion-aware floorplanning, and enhanced DRC/LVS handling, to better approximate real-world design workflows. **Fourth**, we will **enhance the general capabilities** of our agentic platform by incorporating modules currently absent from the framework, such as long-term memory and self-improvement mechanisms that enable the agent to learn from past successes and failures. **Finally**, we plan to improve **evaluation fidelity** by incorporating additional signoff metrics and richer feedback signals, further bridging the gap between benchmark settings and production environments.

A.3 CHARACTERISTICS OF PDAGENT-BENCH

We summarize the key characteristics of each subset in Figures 4–7, including question count, difficulty distribution, topic coverage, and evaluation-rubric granularity. Together, the four subsets comprise 343 questions spanning the full physical-design workflow, from foundational concepts to tool-specific script generation.

A.4 BENCHMARK RESULTS FOR ALL EVALUATED MODELS

Table 6 merges the results of the earlier evaluation (eleven models) with those of the five recent models reported in Table 4. The rule-graded tasks use the same checkers in both evaluations and can be compared directly. The judge-scored tasks (Foundational, RootOpt, Report) were graded differently in the two evaluations, so they should be compared within each block only.

Progress across evaluation rounds. On the rule-graded tasks, the five recent models improve only marginally over the strongest earlier models. The best ICC2 pass@1 rises from 45.5% (GPT-5.5) to 52.3% (GPT-5.6 Sol), whereas the best Innovus pass@1 remains flat (42.2% vs. 41.1%), and the best Innovus pass@5 is identical in both rounds (51.1%). The best STA and FM pass@1 are also unchanged (72.7% and 81.8%). The only higher best score is on ECO (60.0% for GPT-5.6 Luna vs. 50.0%), a difference of a single task out of ten. Newer versions within the same model family do not consistently outperform their predecessors: GPT-5.6 Sol matches GPT-5.5 on STA and FM, gains on ICC2, and drops slightly on Innovus and ECO; Claude Opus 5 improves over Claude Opus 4.7 on FM,

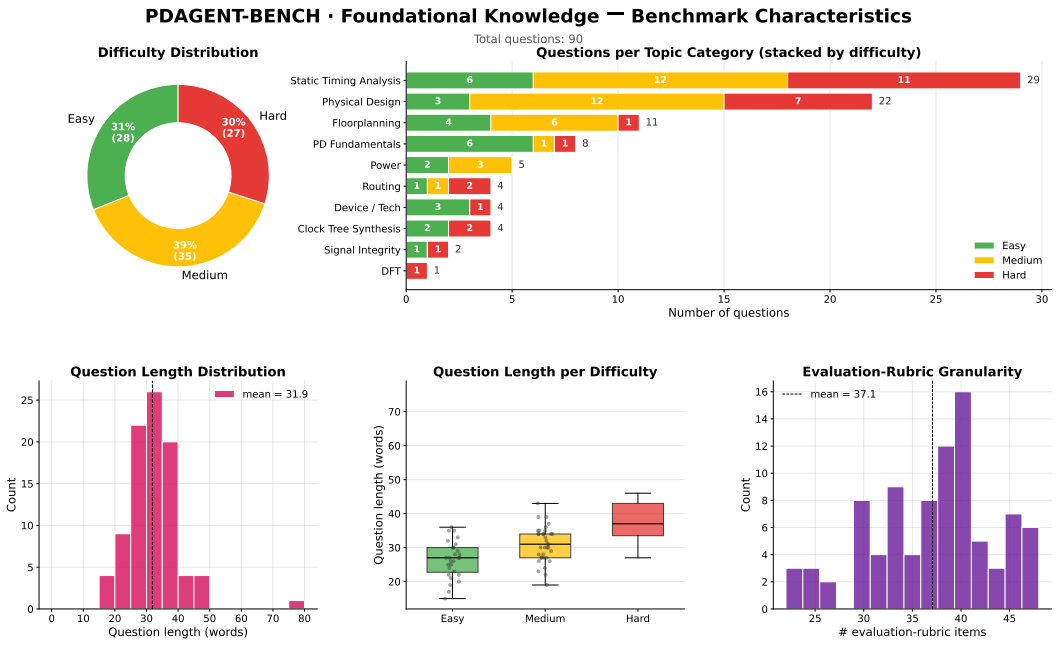


Figure 4: Characteristics of the foundational-knowledge subset of PDAGENT-BENCH (90 questions). Questions are balanced across three difficulty levels and span ten physical-design topics, with timing analysis and physical design as the most represented categories. Each question is paired with a fine-grained evaluation rubric (mean of 37.1 items).

Table 6: Benchmark results for all evaluated models. Each cell reports pass@1 / pass@5. The top block is the earlier evaluation; values in parentheses are the corresponding *human-evaluation* scores, available for Foundational, RootOpt, and Report. The bottom block is the later evaluation of five recent models (Table 4). The rule-graded tasks (STA, FM, ECO, ICC2, Innovus) use the same checkers in both blocks. Foundational, RootOpt, and Report in the bottom block are scored by a blind GPT-5.4-mini judge with pass@5 as the mean over five samples, which differs from the grading of the top block, so these three columns should be compared within a block only. The two blocks also differ in generation settings (the bottom block uses medium reasoning effort and a 32K-token output cap). **Bold** / underline mark the best / second-best result per column *within each block*. N/A: model does not support multi-modal input.

Model	Foundational		RootOpt		Report		STA		FM		ECO		ICC2		Innovus	
	p@1	p@5	p@1	p@5	p@1	p@5	p@1	p@5	p@1	p@5	p@1	p@5	p@1	p@5	p@1	p@5
Earlier evaluation																
Qwen-VL-8B	.533 (.520)	.573 (.558)	.245 (.228)	.319 (.269)	.327 (.293)	.464 (.421)	.545	.636	.273	.455	.100	.100	.068	.102	.122	.133
Qwen-VL-30B	.692 (.661)	.695 (.679)	.379 (.368)	.460 (.400)	.323 (.294)	.427 (.415)	.545	.636	.273	.545	.100	.100	.091	.125	.111	.178
Qwen3-8B	.543 (.488)	.585 (.573)	.255 (.216)	.286 (.263)	N/A		N/A		.273	.500	.000	.000	.080	.102	.122	.144
DeepSeek-Chat	.726 (.689)	.799 (.773)	.486 (.434)	.521 (.473)	N/A		N/A		.409	.591	.300	.500	.340	.364	.256	.356
DeepSeek-Reasoner	.737 (.684)	.806 (.788)	.612 (.585)	.680 (.630)	N/A		N/A		.455	.591	.300	.500	.216	.352	.233	.356
GPT-5-mini	.800 (.701)	.820 (.790)	.560 (.540)	.655 (.615)	.655 (.607)	.685 (.666)	.545	.636	.227	.364	.300	.300	.057	.114	.133	.189
GPT-5.1	.820 (.802)	.840 (.830)	.655 (.619)	.674 (.624)	.655 (.604)	.705 (.649)	.545	.545	.545	.636	.200	.200	.318	.432	.278	.311
Claude Sonnet 4.6	**.826** (.794)	.835 (.821)	.688 (.660)	.700 (.664)	.668 (.625)	.736 (.717)	.636	.636	.636	.682	**.500**	.500	.341	.352	.289	.311
Claude Opus 4.7	.824 (.784)	**.842** (.829)	**.762** (.714)	.767 (.731)	.641 (.614)	.659 (.629)	**.727**	**.727**	.727	.727	.400	.500	.341	.375	.367	.411
GPT-5.5	.804 (.778)	.830 (.807)	.733 (.689)	**.774** (.732)	**.746** (.720)	**.750** (.732)	**.727**	**.727**	**.818**	**.864**	**.500**	**.600**	**.455**	**.580**	**.422**	**.511**
GPT-5.4-mini	.723 (.668)	.741 (.726)	.545 (.502)	.581 (.526)	.461 (.422)	.514 (.487)	.636	.636	.591	.727	**.500**	.500	.295	.398	.267	.356
Later evaluation																
DeepSeek-V4.1-Flash	.848	.863	.669	.659	.505	.566	**.727**	**.727**	.773	**.955**	.400	.500	.364	.466	.322	.444
GPT-5.6 Luna	.848	.829	.576	.590	.600	.604	.455	.545	.591	.773	**.600**	.500	.386	.455	.367	.467
GPT-5.6 Sol	.857	.871	.657	.671	**.682**	.658	**.727**	.636	**.818**	.818	.400	.400	**.523**	**.602**	**.411**	**.511**
Claude Sonnet 5	.829	.832	.574	.579	.623	.608	.545	**.727**	.545	.636	.400	**.600**	.273	.341	.333	.367
Claude Opus 5	**.900**	**.908**	**.748**	**.785**	.536	**.684**	.636	**.727**	**.818**	.909	.400	.400	.398	.534	.400	.444

ICC2, and Innovus but drops on STA; and Claude Sonnet 5 scores below Claude Sonnet 4.6 on four

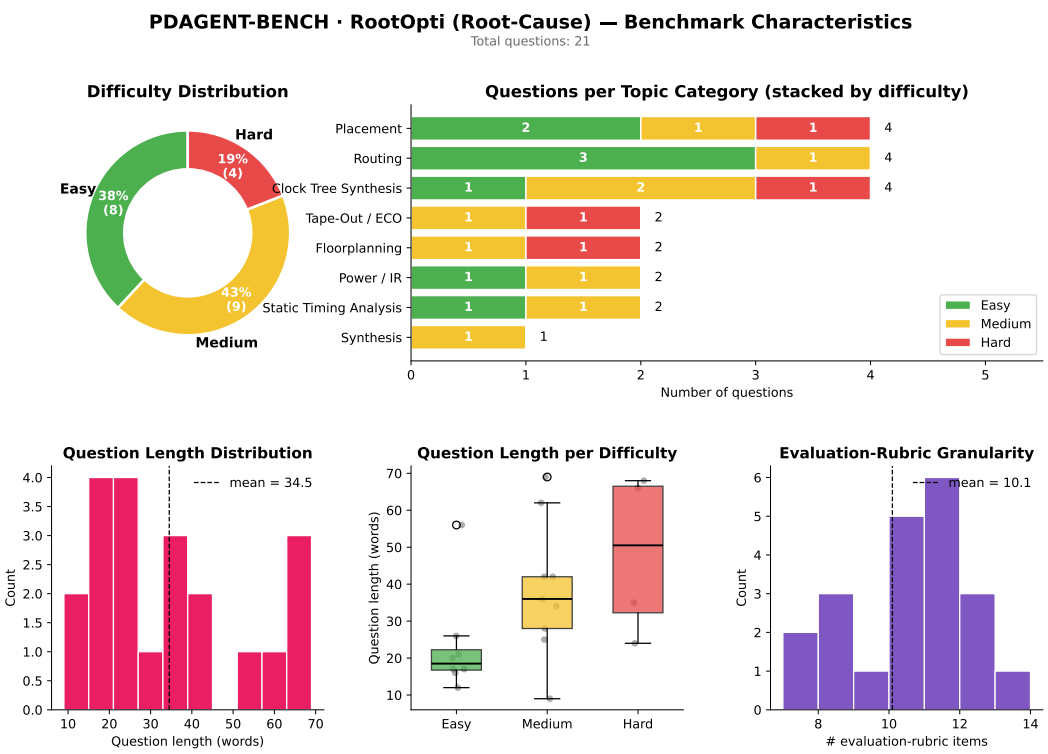


Figure 5: Characteristics of the root-cause analysis subset of PDAGENT-BENCH (21 questions). Questions span eight physical-design stages, from synthesis through tape-out/ECO, with placement, routing, and clock tree synthesis as the most represented categories. Each question is paired with a targeted evaluation rubric (mean of 10.1 items).

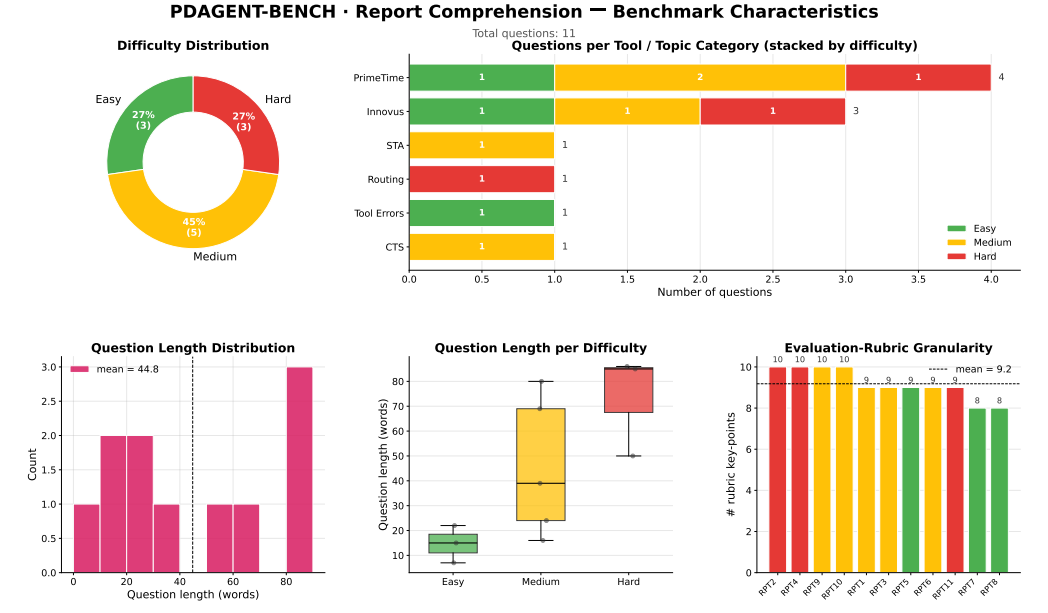


Figure 6: Characteristics of the report-comprehension subset of PDAGENT-BENCH (11 questions). Questions cover reports from major EDA tools (PrimeTime, Innovus) across stages including STA, routing, CTS, and tool error logs. Each report is paired with a structured evaluation rubric.

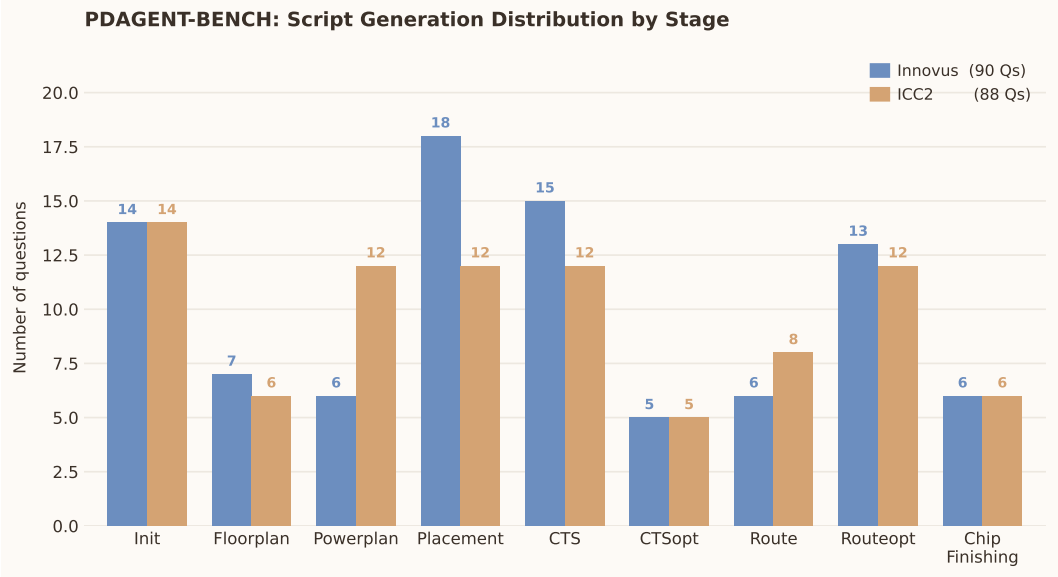


Figure 7: Characteristics of the script generation subset of PDAGENT-BENCH (178 questions in total). Questions span nine physical-design stages, from initialization through chip finishing, and are provided in parallel for two industrial EDA tools, Innovus (90 questions) and ICC2 (88 questions). Placement, CTS, and routing-optimization stages account for the largest share of questions in both tools.

of the five rule-graded tasks. Because the two rounds also differ in generation settings (Section **??**), these family-level differences should be read as indicative rather than as controlled comparisons.

Open-source models. DeepSeek-V4.1-Flash substantially narrows the gap between open-source and proprietary models. Unlike the earlier DeepSeek models, it accepts image inputs and can therefore be evaluated on Report and STA, where it ties for the best STA score (72.7%). On FM, the gap to the best proprietary model shrinks from 36.3 points in the earlier round (DeepSeek-Reasoner, 45.5%, vs. GPT-5.5, 81.8%) to 4.5 points (77.3% vs. 81.8%), and its FM pass@5 (95.5%) is the highest of all sixteen models. On Innovus, the gap shrinks from 16.6 to 8.9 points, but on ICC2 it widens from 11.5 to 15.9 points. This progress comes at a higher inference cost: across the benchmark, DeepSeek-V4.1-Flash produces 3–6$\times$ more output tokens than the other recent models.

Understanding tasks. Within the recent block, all five models score between 82.9% and 90.0% on Foundational, suggesting that textbook-level physical design knowledge is close to saturation, whereas RootOpt spans a wider range (57.4%–74.8%) and continues to separate models by their diagnostic reasoning. Report, STA, and ECO contain only 10–11 tasks each, so a single task changes the score by 9–10 points. Because pass@5 is computed from five samples that exclude the pass@1 sample (and is averaged over the five samples for judge-scored tasks), it can fall below pass@1, most visibly on these small subsets (e.g., GPT-5.6 Sol on STA).

A.5 Error Case Analysis

To understand the failure modes of current LLMs on physical-design tasks, we manually inspect the incorrect outputs of all evaluated models, categorize them by error type, and report the resulting distribution. The analysis surfaces the systematic weaknesses underlying the quantitative gaps reported in Table 4 and informs the directions discussed in Section 4.3.

A.6 Results on Full Flow Implementation

The evaluation results of **PDAgent** on the PDAgent-Bench holistic flow suite are summarized in Table 10, using GPT-5mini as the backbone model. The results demonstrate that **PDAgent** is capable of executing a holistic physical-design flow end-to-end, performing self-iterative design

Table 7: Failure breakdown of ChatGPT-5.5 on `benchmark_innovus.json` (52 failures out of 90).

Category	Count	Description
Multi-line near-miss	33	Right intent/commands, wrong flag names or missing/extra steps
Same command, wrong attribute	12	Single-line command correct, but flag is wrong
Same command, value formatting	3	Right command + flags, formatting/value diff
Wrong command	4	Picked wrong command (incl. 1 free-text answer)
Total	**52**	

Table 8: Examples of attribute-name hallucinations: the model picks the correct command but invents plausible-but-wrong Innovus flag names.

ID	Expected flag	Generated flag
TC_016	`-around core -center 1`	`-follow core`
TC_018	`-subclass`	`-userDefined`
TC_021	`-no_routing_blkg`	`-ignore_routing_blockage`
TC_026	`-rule ... -wellCutCell`	`-distance ... -boundaryCell`
TC_038	`-place_opt_save_db`	`-place_global_save_db`
TC_054	`-outDir`	`-report_dir`
TC_069	`-drouteMinSlackForWireOptimization`	`-droutePostRouteSpreadWireMinSlack`
TC_077	`lefDefOutVersion`	`defOutVersion`
TC_084	`specifyCellPad CLK*` (positional)	`specifyCellPad -cell CLK*`

space exploration, autonomously resolving DRC violations, and optimizing the design toward timing closure. The consistent performance across diverse designs demonstrates the generality of our method. On average, **PDAgent** completes the full flow through 53 interactions with the EDA tools, reflecting the iterative nature of physical-design optimization.

Our method performs well on small and medium-complexity designs, achieving timing closure with zero or minimal DRC violations at the final RouteOpt stage. However, on larger and more complex designs such as NVDLA-large and Open910, residual DRC violations remain after RouteOpt, and timing closure is not fully achieved in all cases. We attribute these limitations not only to the inherent complexity of the design space at advanced technology nodes, but also to the current lack of more sophisticated ECO and DRC-fixing capabilities in our framework, which we identify as a promising direction for future work. We anticipate that future work building upon **PDAgent** can achieve improved timing slack, higher utilization, cleaner DRC results, and reduced interaction overhead through more advanced optimization strategies.

Comparison Against RAG-enhanced Agent Baseline

We further evaluate two RAG-enhanced LLM baselines, which retrieve relevant content from EDA command reference manuals and physical-design guidebooks, using GPT-5mini backbone models for fair comparison.The Full flow suite results are reported in Table 11.

For the Full flow suite, the RAG-enhanced method fails to complete most designs beyond the first two stages. We attribute this performance gap to the highly structured and interdependent nature of physical-design flows: each stage comprises multiple steps with precise tool-level syntax and context-specific requirements, which general-purpose retrieved documentation is too coarse-grained to fully capture. Unlike **PDAgent**, which encodes actionable, stage-specific skills, the RAG-based approach lacks the procedural knowledge necessary to handle tool-specific errors, adapt parameters across designs, and drive the flow to completion.

Ablation: LLM Without Skills or Retrieval

To isolate the contribution of the LLM from that of the skill library, we run GPT-5-mini alone, through the bare API without the skill library or retrieval, on all ten designs of the full-flow suite in Cadence Innovus. Table 12 reports the results. The model fails at initialization on every design, so no design completes initialization, floorplanning, and power planning. The failures are grounding errors rather than reasoning errors: the model invokes commands that do not exist in Innovus (e.g., `read_lef`, `pdnInit`, `createPowerRing`, `getDieArea`), passes illegal options to valid commands (e.g., `setDesignMode`), or issues Stylus commands such as `create_floorplan` in a legacy-mode session. This is consistent with the script-generation results: GPT-5-mini reaches only 13.3% pass@1 on Innovus script generation (Table 6), and most script-generation failures are near-misses with

Table 9: GPT-5.5 Basic Benchmark Executive Summary (90 questions).

Overall Score: 724.0 / 900.5 = 80.4%	
By Difficulty	Easy 83.0% | Medium 81.7% | Hard 76.0%
Catastrophic Fails (<60%)	2 / 90 ($\rightarrow$ Q2: 15%, Q86: 45%)
Low Criteria ($\leq$50% of max)	32 / 454 ($\sim$7% — concentrated, not diffuse)
Strengths vs. Weaknesses	
Strengths	• File formats (GDSII / SDF / DEF, all $\geq$85%) • STA definitions (slack, false path, best/worst path: 100%) • Technology-node comparison (Q63: 100%) • Digital logic fundamentals • Physical Design Fundamentals topic family (89%)
Weaknesses	• Question-interpretation errors (Q2) • Quantitative reasoning / layer math (Q86) • Library models — NLDM vs. CCS (65%) • CTS insertion-delay tuning • Post-route to signoff correlation
Top 3 Error Patterns	
1	Misreading the question's framing $\rightarrow$ wrong taxonomy (Q2).
2	Quantitative reasoning with constraint propagation (Q86).
3	"Missing one column" — partial trade-off tables, lost bonus points.
Verdict	
Strong general PD/STA knowledge; points are lost to under-elaboration and one-off interpretation errors, not deep knowledge gaps.	

Table 10: Full-flow implementation results of **PDAgent** on the PDAgent-Bench full flow suite (TSMC 28nm). WNS and DRC violations are reported at each stage; positive WNS indicates timing closure. Utilization (Util.) is reported after floorplanning.

Design	init	floorplan	powerplan	place	CTS	CTSopt	Route	Routeopt
TinyRISCV	5ns, WNS: 1.362ns Setup successfully	Util: 0.65 DRC Vio: 0	DRC Vio: 0	WNS : -0.014ns DRC Vio: 0 Density: 0.715	WNS : -0.127ns DRC Vio: 0 Density: 0.7165	WNS : 0.001ns DRC Vio: 0 Density: 0.713	WNS : 0.021ns DRC Vio: 1 Density: 0.722	WNS : 0.021ns DRC Vio: 0 Density: 0.722
Open910	1.5ns, WNS: 0.465ns Setup successfully	Util: 0.60 DRC Vio: 0	DRC Vio: 0	WNS : -0.05ns DRC Vio: 55 Density: 0.584	WNS : -0.009ns DRC Vio: 2 Density: 0.653	WNS : 0.001ns DRC Vio: 0 Density: 0.654	WNS : -0.431ns DRC Vio: 127 Density: 0.703	WNS : 0.008ns DRC Vio: 0 Density: 0.731
Systolic array	3.4ns, WNS: 1.11ns Setup successfully	Util: 0.5 DRC Vio: 0	DRC Vio: 0	WNS : 0.013ns DRC Vio: 0 Density: 0.464	WNS : 0.003ns DRC Vio: 0 Density: 0.464	WNS : 0.003ns DRC Vio: 0 Density: 0.464	WNS : -0.312ns DRC Vio: 0 Density: 0.464	WNS : 0.002ns DRC Vio: 0 Density: 0.466
16b pipelined mult.	2.6ns, WNS: 0.808ns Setup successfully	Util: 0.55 DRC Vio: 0	DRC Vio: 0	WNS : 0.02ns DRC Vio: 0 Density: 0.625	WNS : -0.008ns DRC Vio: 0 Density: 0.6289	WNS : 0.012ns DRC Vio: 0 Density: 0.629	WNS : 0.251ns DRC Vio: 0 Density: 0.629	WNS : 0.251ns DRC Vio: 0 Density: 0.629
AES-256	3.5ns, WNS: 1.003ns Setup successfully	Util: 0.60 DRC Vio: 0	DRC Vio: 0	WNS : 0.003ns DRC Vio: 0 Density: 0.613	WNS : 0.006ns DRC Vio: 0 Density: 0.615	WNS : 0.006ns DRC Vio: 0 Density: 0.615	WNS : 0.12ns DRC Vio: 303 Density: 0.615	WNS : 0.119ns DRC Vio: 0 Density: 0.615
NVDLA-small	2ns, WNS: 0.65ns Setup successfully	Util: 0.60 DRC Vio: 0	DRC Vio: 0	WNS : -1.06ns DRC Vio: 46 Density: 0.424	WNS : 0.002ns DRC Vio: 0 Density: 0.554	WNS : 0.002ns DRC Vio: 0 Density: 0.554	WNS : -0.005ns DRC Vio: 21 Density: 0.556	WNS : 0.002ns DRC Vio: 0 Density: 0.556
NVDLA-large	2.5ns, WNS: 0.69ns Setup successfully	Util: 0.60 DRC Vio: 0	DRC Vio: 0	WNS : -0.024ns DRC Vio: 68 Density: 0.487	WNS : -0.145ns DRC Vio: 0 Density: 0.586	WNS : -0.063ns DRC Vio: 0 Density: 0.594	WNS : 0.021ns DRC Vio: 242 Density: 0.596	WNS : 0.021ns DRC Vio: 0 Density: 0.598
UART controller	3.4ns, WNS: 1.108ns Setup successfully	Util: 0.65 DRC Vio: 0	DRC Vio: 0	WNS : 1.1ns DRC Vio: 0 Density: 0.782	WNS : 1.4ns DRC Vio: 0 Density: 0.789	WNS : 1.59ns DRC Vio: 0 Density: 0.789	WNS : -0.445ns DRC Vio: 0 Density: 0.85	WNS : -0.09ns DRC Vio: 0 Density: 0.85
Elevator FSM	2.5ns, WNS: 0.708ns Setup successfully	Util: 0.7 DRC Vio: 0	DRC Vio: 0	WNS : 0.703ns DRC Vio: 2 Density: 0.664	WNS : 0.702ns DRC Vio: 2 Density: 0.665	WNS : 0.703ns DRC Vio: 0 Density: 0.667	WNS : 0.622ns DRC Vio: 0 Density: 0.667	WNS : 0.622ns DRC Vio: 0 Density: 0.667
Ethernet MAC	10ns, WNS: 5.621ns Setup successfully	Util: 0.60 DRC Vio: 0	DRC Vio: 0	WNS : 5.62ns DRC Vio: 25 Density: 0.62	WNS : 5.475ns DRC Vio: 23 Density: 0.624	WNS : 5.470ns DRC Vio: 0 Density: 0.628	WNS : 5.53ns DRC Vio: 69 Density: 0.624	WNS : 5.53ns DRC Vio: 0 Density: 0.625

the correct intent but incorrect syntax (Finding 1 in Section 4.3). Because the early stages alone require a long sequence of correct commands, these per-command errors compound, and end-to-end success is out of reach without grounding in tool-specific syntax. Together with the RAG baseline

Table 11: Early-stage physical design results using a RAG-based method without skills on the PDAgent-Bench full flow suite (TSMC 28nm). Results cover the first three implementation stages: synthesis, floorplanning, and power planning. The RAG approach exhibits consistent failures across all stages, including setup errors, pin placement failures, and incomplete power networks, demonstrating its limited capability in automated physical design.

Design	init	floorplan	powerplan
TinyRISCV	5ns, WNS: 1.362ns Setup failed	Pin placement failed	Failed to use the correct layer
Open910	1.5ns, WNS: 0.465ns Setup failed	Pin placement failed	Failed to addRing
Systolic array	3.4ns, WNS: 1.11ns Setup successfully	Pin placement failed	DRC Vio: 346
16b pipelined mult.	2.6ns, WNS: 0.808ns Setup failed	Util: 0.65 DRC Vio: 0	Missing Bottom and Top Power Rings
AES-256	3.5ns, WNS: 1.003ns Setup failed	Pin placement failed	Stripes Not Connected to Power Ring
NVDLA-small	2ns, WNS: 0.65ns Setup failed	Pin placement failed	Missing Bottom and Top Power Rings
NVDLA-large	2.5ns, WNS: 0.69ns Setup failed	Pin placement failed	Missing Bottom and Top Power Rings
UART controller	3.4ns, WNS: 1.108ns Setup successfully	Util: 0.65 DRC Vio: 0	Stripes Not Connected to Power Ring
Elevator FSM	2.5ns, WNS: 0.708ns Setup successfully	Util: 0.7 DRC Vio: 0	Stripes Not Connected to Power Ring
Ethernet MAC	10ns, WNS: 5.621ns Setup failed	Pin placement failed	Missing Bottom and Top Power Rings

Table 12: Early-stage physical design with GPT-5-mini alone (bare API, no skill library, no retrieval) on the full-flow suite in Cadence Innovus. Each cell lists the error that caused the stage to fail: initialization did not complete, the floorplan was not created, or no power net was built. Because no design passed initialization, the floorplan and power-plan columns report the failures observed when these stages were attempted under the same protocol.

Design	Init	Floorplan	Power plan
TinyRISCV	Illegal options on `setDesignMode`, `create_rc_corner`	Illegal option on `setDesignMode`	Invalid command `getDesignCoreArea`
Open910	Illegal options on `setDesignMode`, `create_rc_corner`	Illegal option on `setDesignMode`	Ambiguous command `getNet`
Systolic array	Invalid command `read_lef`	Stylus `create_floorplan` in legacy session	Invalid command `create_net`
16b pipelined mult.	Invalid command `read_lef`	Stylus `create_floorplan` in legacy session	Ambiguous command `set_design`
AES-256	Invalid commands `read_liberty`, `read_lef`	Stylus `create_floorplan` in legacy session	Ambiguous command `getNet`
NVDLA-small	Invalid command `read_lef`	Invalid command `createFloorplan`	Illegal option on `set_power_network_optimization_mode`
NVDLA-large	Invalid command `read_lef`	Stylus `create_floorplan` in legacy session	Invalid command `pdnInit`
UART controller	Nonexistent MMMC package `mmc`	Stylus `create_floorplan` in legacy session	Invalid command `getDieArea`
Elevator FSM	Invalid commands `create_lib_set`, `mmc::create_lib_set`	Stylus `create_floorplan` in legacy session	Invalid command `pdnCreate`
Ethernet MAC	Invalid commands `read_liberty`, `create_libset`	Stylus `create_floorplan` in legacy session	Invalid command `createPowerRing`

(Table 11) and the full agent (Table 10), this ablation clarifies the division of labor in **PDAgent**: the skill library supplies the tool-specific command vocabulary that the model lacks, while the LLM remains responsible for stage sequencing, parameter selection, and error recovery.

A.7 Token Usage of full flow implementation

Table 13 reports token consumption for the full flow across the ten benchmarks. The cost is overwhelmingly dominated by input (context) tokens: 5.84M. This is expected for an agentic EDA loop, in which each step ingests large synthesis logs, timing reports, and netlist summaries but emits only short commands and decisions. Token usage tracks design complexity: NVDLA-large, the largest design (1.48M cells), is the most expensive at 1.62M tokens and 192 calls, whereas the small control designs (UART, Elevator FSM, 16b multiplier) converge in 27–34 calls and under 180k tokens each.

A.8 Macro Placement results

We evaluate on five ChiPBench designs against seven learning-based and analytical placers (WireMask-EA, SA, MaskPlace, ChiPFormer, DREAMPlace, AutoDMP, and OpenROAD) to show that our platform can handle the complex macro placement. For each design, the model generates five candidate placements that are refined in parallel, with each candidate given up to five rounds

Table 13: Token usage of the full-flow implementation across all benchmark designs. We report input tokens, output tokens, the resulting total, and the number of LLM calls.

Design	Input	Output	Total	LLM calls
TinyRISCV	327,648	6,013	333,661	67
Open910	1,283,641	24,167	1,307,808	182
Systolic array	227,648	4,013	231,661	47
16b pipelined mult.	144,057	3,924	147,981	29
AES-256	259,783	3,948	263,731	57
NVDLA-small	1,259,783	21,948	1,281,731	137
NVDLA-large	1,595,980	25,943	1,621,923	192
UART controller	127,648	4,013	131,661	27
Elevator FSM	174,057	4,524	178,581	34
Ethernet MAC	444,057	8,924	452,981	79

of feedback-driven refinement. As shown in Table 14, our flow achieves the best WNS and TNS on all four multi-baseline designs, cutting TNS by up to $\sim 60\times$ over the strongest baseline (e.g., $-244.94 \rightarrow -4.01$ on ariane133) and staying on par on swerv_wrapper. It also attains the lowest power, at the cost of larger area. On VeriGPU, where OpenROAD is the only baseline, it reaches a comparable point with better TNS. Overall, an 8B vision-language backbone driven by our flow can match or surpass specialized macro placers on timing closure.

A.9 PDAgent Design Details

A.9.1 Agent Design

Each agent inherits from a common `BaseAgent` class that provides message handling, LLM access, tool query primitives, and a shared `DesignContext`. The `DesignContext` is a cached representation of the active design, populated by querying the EDA tool after initialization, including macro locations, die/core area, and clock topology. It is shared across all agents to maintain consistent design state throughout the optimization loop.

Worker Agent. The Worker Agent maintains a persistent EDA session and executes subtasks in three modes: (1) skill-guided generation, where the LLM synthesizes Tcl scripts from skill content and design context; (2) script execution, where reference Tcl scripts are sourced and run directly; and (3) optimization application, where targeted optimizations requested by the Optimizer Agent are applied and coordinated through the Planner Agent. Executed scripts, logs, and layout snapshots are organized by stage in a session history with manifest tracking, ensuring full traceability and enabling downstream agents to reference artifacts from any prior stage.

Analyzer Agent. The Analyzer Agent is selectively activated at analysis-critical stages (e.g., placement, CTS, and routing), where raw EDA tool logs are verbose and difficult to interpret directly. At each activated stage, it extracts a structured `StageMetrics` vector covering timing, utilization, routing, clock, and power metrics, and passes this condensed summary to the Debugger and Optimizer Agents for closed-loop refinement. Lightweight stages such as floorplanning and powerplanning bypass analysis, reducing runtime overhead without sacrificing result quality at critical design junctures.

Debugger Agent. After the Worker Agent completes a stage, the Debugger Agent checks for DRC violations and other errors by executing an iterative diagnosis loop. It loads the stage-specific debug skill, initializes a multi-turn LLM conversation with the skill content, EDA output, and convergence state as context, then iteratively performs check→fix→check cycles via direct tool queries. Each step is tracked in an `AgentStageContext` that maintains the full conversation history, violations found and fixed, and budget consumption. Skills provide the check criteria and corresponding fix strategies for each stage.

To prevent unbounded iteration, a three-tier budget hierarchy is enforced: (1) micro-level, handling syntax errors with up to 2 retries; (2) meso-level, handling parameter adjustments with up to 4 retries; and (3) macro-level, handling cross-stage rollback with up to 2 retries. When the budget is exhausted, the Debugger Agent escalates to the Optimizer Agent for more aggressive intervention.

Table 14: Macro placement results across all benchmark designs (our method as *Ours*).

Design	Method	Power ↓	WNS ↑	TNS ↑	Area ↓
ariane133	WireMask-EA	0.369	-0.417	-329.353	349228
	SA	0.365	-0.512	-650.399	347043
	MaskPlace	0.373	-0.349	-244.936	357322
	ChiPFormer	0.370	-0.553	-860.952	349005
	DREAMPlace	0.367	-0.540	-690.266	348180
	AutoDMP	0.350	-0.982	-1913.660	**344091**
	OpenROAD	0.359	-0.498	-596.718	352706
	Ours	**0.257**	**-0.08**	**-4.01**	357917
bp_be	WireMask-EA	0.467	-2.142	-4093.970	123966
	SA	0.459	-2.494	-5510.140	124667
	MaskPlace	0.456	-2.439	-4459.870	124938
	ChiPFormer	0.425	-2.169	-3541.820	**110904**
	DREAMPlace	0.458	-2.163	-3648.020	125221
	AutoDMP	0.453	-1.948	-3351.120	125471
	OpenROAD	0.409	-1.862	-2343.290	118786
	Ours	**0.421**	**-0.63**	**-57.67**	128865
bp_fe	WireMask-EA	0.309	-1.666	-777.420	77648
	SA	0.310	-1.179	-1236.880	81919
	MaskPlace	0.309	-1.318	-771.363	77881
	ChiPFormer	0.299	-1.197	-1000.190	**72723**
	DREAMPlace	0.294	-1.117	-473.261	77427
	AutoDMP	0.312	-1.253	-1198.430	81294
	OpenROAD	0.285	-1.092	-491.698	75884
	Ours	**0.24**	**-0.43**	**-29.06**	81323
swerv_wrapper	WireMask-EA	0.666	-1.027	-873.506	205627
	SA	0.648	-0.962	-854.737	**200280**
	MaskPlace	0.690	-1.251	-1306.460	206481
	ChiPFormer	0.674	-1.189	-1282.240	205775
	DREAMPlace	0.646	-1.061	-780.196	203896
	AutoDMP	0.648	-1.094	-773.339	200823
	OpenROAD	0.628	-1.127	-1163.820	202853
	Ours	**0.626**	**-0.67**	**-750.37**	204713
VeriGPU	OpenROAD	**0.0779**	**-0.26**	-16.84	**271391**
	Ours	0.0931	-0.28	**-14.14**	273414

Optimizer Agent. The Optimizer Agent operates in two modes. In iterative mode, it conducts a multi-turn LLM-guided loop analogous to the Debugger Agent: checking current metrics, adjusting parameters (e.g., placement effort illustrated in Figure 2(c)), informed by stage-specific optimization skills and the full iteration trajectory stored in `MetricsDB`, then verifying improvements. In tree-of-thought mode, triggered on escalation or LLM-directed exploration, it performs branching design space exploration by generating multiple solution candidates and executing them either in parallel or sequentially, as described in §A.9.2.

A.9.2 Closed-Loop PPA Optimization Mechanism

The PPA Optimization Mechanism in PDAgent operates at two levels: a **stage-level** that progresses through the physical design flow step by step, and a **parameter-level** that explores the design space within each stage.

Stage-Level Execution. The Planning Agent maintains a plan $\mathcal{P} = \{s_1, \ldots, s_n\}$ of subtasks ordered by dependency. For each subtask s_i, execution follows one of two paths based on the tool output:

(1) **Success path:** Tool $\xrightarrow{\text{result}}$ Planner $\xrightarrow{\text{if worthy}}$ Analyzer $\xrightarrow{\text{metrics}}$ Optimizer $\xrightarrow{\text{decision}}$ Planner $\rightarrow$ advance or re-run;

> You are a senior VLSI physical design engineer. Answer the user's question with technically accurate, well-structured explanations. Cover key concepts, trade-offs, and any relevant tool/flow details. Be concise, compact but complete.

Figure 8: System prompt used for Physical Design Understanding tasks.

(2) **Failure path:** Tool $\xrightarrow{\text{error}}$ Planner $\rightarrow$ Debugger $\xrightarrow{\text{action}}$ Planner $\rightarrow$ {retry, fix, rollback, skip, escalate}. The convergence state, classified as improving, stalled, or diverging based on the metric trajectory across iterations, informs action selection at each decision point.

Parameter-Level Exploration. When the Optimizer Agent determines that local parameter adjustments are insufficient, for example upon escalation from the Debugger Agent or detection of a stalled convergence state, it activates a tree-of-thought search over the parameter space. The search proceeds in batches: at each batch $b \in \{1, \ldots, B\}$, the LLM generates K candidate parameter configurations $\{c_1^{(b)}, \ldots, c_K^{(b)}\}$, such as redoing macro placement or revising the power plan to reduce routing layer pressure, each accompanied by the corresponding EDA commands and evaluation queries. Each candidate is evaluated by saving a design checkpoint, applying the parameter modifications, executing the evaluation commands, and extracting the resulting metrics. The design is then restored to the checkpoint before the next candidate is evaluated.

The fitness of each candidate is computed as a stage-aware weighted score:

$$\text{score}(c) = \sum_{m \in \mathcal{M}_s} w_m^{(s)} \cdot \hat{v}_m(c), \tag{1}$$

where $\mathcal{M}_s$ denotes the metric set for stage s, $w_m^{(s)}$ the stage-specific weight, and $\hat{v}_m(c) \in [0, 1]$ the normalized metric value. The best candidate $c^* = \arg\max_c \text{score}(c)$ becomes the starting point for batch $b{+}1$. The search terminates when stage requirements are satisfied (e.g., DRC count $= 0$ for routing), the score improvement falls below threshold δ, or the batch budget B is exhausted.

Iteration Tracking. A persistent `MetricsDB` records the full parameter to metric mapping across all iterations, including parameter provenance, capturing which agent modified which parameter and the rationale behind each change. This trajectory data serves two purposes: it provides the Optimizer Agent with historical context for informed tuning decisions, and it enables post-hoc analysis of the DSO process for reproducibility and insight.

A.10 System Prompts

We list the system and judge prompts used throughout our evaluation in Figures 8–10. Figure 8 shows the system prompt provided to all evaluated models on the Physical Design Understanding tasks (Basic, RootOpt, Report, STA). Figure 9 shows the system prompt for the script-generation tasks (Innovus, ICC2, ECO, FM). Figure 10 shows the prompt given to our LLM-Judge (Claude Sonnet 4.6) for automatic scoring of free-response answers.

> You are a senior VLSI physical design engineer. Answer each question with ONLY the exact Tcl/shell command(s) required. Do not add explanations, comments, markdown fences, or surrounding prose.

Figure 9: System prompt used at inference time for the script Generation tasks (Innovus, ICC2, ECO, FM).

A.11 Examples from PDAgent-Bench

We illustrate the structure and content of PDAgent-Bench through representative examples drawn from each task category. Figures 11–12 show a sample question, reference answer, and scoring rubric

Please act as an impartial physical-design engineer and judge the quality of the response provided by an AI assistant to the user question shown below. Evaluate the response objectively against the rubric supplied with each data item.
Begin your evaluation with a brief explanation justifying your assessment. Be as fair and objective as possible.
After the explanation, conclude with a numerical score from 0 to 10 in the following strict format:
score: X
For example: **score: 7**

Figure 10: LLM-Judge prompt used to score model responses on the Physical Design Understanding tasks.

from the Basic benchmark. The complete benchmark, along with all model outputs and scoring results, is included in the supplementary material.

Question. In VLSI backend physical design flow, describe the essential input files required by a physical design tool and the key output files it generates. For each file, briefly explain its purpose and common format extensions.

Reference Answer.
Input data required for physical design:

1. **Technology file** (`.tf` in Synopsys, `.techlef` in Cadence): describes units, drawing patterns, layers, design rules, vias, and parasitic R/C of the manufacturing process.
2. **Physical libraries** (`.lef`, `.gds`; or Synopsys `.CEL`, `.FRAM`): layout information and abstract models for placement and routing (pin accessibility, blockages, etc.).
3. **Timing, logical, and power libraries** (`.lib`, or LM view `.db`): timing and power information for all design elements.
4. **TDF file** (`.tdf` / `.io`): pad/pin arrangement (order and location); for full-chip flows also captures VDD/VSS pads and power-cut diodes not present in the Verilog netlist.
5. **Constraints** (`.sdc`): area, power, and timing constraints.
6. **PDEF** (optional): row and cell placement locations.
7. **DEF** (optional): row, cell, and pre-existing placement information.

Output data from a physical design tool:

1. **Standard Delay Format** (`.sdf`): post-layout timing (excluding load information).
2. **Parasitics** (`.spef`, `.dspf`): extracted R/C of cells and nets.
3. **Post-routed netlist** (`.v`; flat or hierarchical): connectivity of all cells.
4. **Physical layout** (`.gds`): final layout for tape-out.
5. **Design Exchange Format** (`.def`): row, cell, and net placement locations.

Figure 11: Example question and reference answer from the Basic benchmark (Physical Design Understanding).

Evaluation Rubric (total: 10 points)

1. **Input Files — Technology & Libraries (3 pts).** 1 pt each for: (a) technology file (`.tf` / `.techlef`) and its role in process rules and parasitics; (b) physical libraries (`.lef` / `.gds` / `.cel` / `.fram`) and their role in layout abstraction; (c) timing/power libraries (`.lib` / `.db`) and their role in timing/power characterization.
2. **Input Files — Constraints & Optional Files (2 pts).** 1 pt each for: (a) SDC constraints file (timing, area, power); (b) IO/TDF or DEF/PDEF for pad placement or pre-existing placement data.
3. **Output Files — Timing & Parasitics (2 pts).** 1 pt each for: (a) SDF file for post-layout timing delays; (b) SPEF or DSPF for extracted RC parasitics.

4. **Output Files — Netlist, Layout & DEF (2 pts).** 1 pt for the post-routed Verilog netlist (`.v`, flat or hierarchical); 0.5 pt for GDS (physical layout); 0.5 pt for DEF (final placement and routing data).
5. **Clarity, Completeness & Technical Accuracy (1 pt).** Answer is well-organized (inputs vs. outputs clearly separated), file extensions correctly associated with descriptions, and no significant factual errors or omissions of major file types.

Figure 12: Scoring rubric corresponding to Figure 11, used by the LLM-Judge to grade model responses.

A.12 Privacy-Preserving Deployment with Multiple Commercial Parties

The agentic PD design workflow involves collaboration among commercial EDA tools, PDKs, and LLMs/VLMs, all of which must remain proprietary. To date, there are still no standards or policies to guide such design workflows. This creates major deployment and operability barriers for the community, as highlighted at the recent first Cognitive Silicon Design Workshop (CSDW) (Workshop). To maximally preserve privacy between these parties, our framework uses a Python coordinator that runs on local hardware/environment and mediates all interactions between commercial EDA tools, PDKs, and foundation models. Rather than allowing an LLM/VLM to invoke tools directly, the coordinator executes each tool command and writes its output to predefined disk locations. A deterministic parser then extracts only task-relevant fields, such as sign-off metrics (WNS, DRC count, and utilization) and compact error signatures. Consequently, the model is never exposed to raw tool dumps, foundry libraries, technology LEF/LIB files, or other proprietary artifacts. It receives only a narrow, sanitizable slice of information explicitly selected by the coordinator. This design makes the privacy boundary an enforceable software interface under our control, rather than relying on the model to prevent the leakage or misuse of sensitive information.