PV-DEBUGGER: AGENTIC DEBUGGING FOR VLSI PHYSICAL VERIFICATION

Anonymous authors
Paper under double-blind review

ABSTRACT

Electronic design automation (EDA) has automated much of the very large-scale integrated circuit (VLSI) design flow, yet physical verification (PV) remains manual. Debugging design rule checking (DRC) violations and layout-versus-schematic (LVS) mismatches in PV requires interpreting reports, localizing faults, repairing layouts, and iteratively re-running sign-off, a challenging workflow largely unexplored by existing agents. We present **PV-Debugger**, a framework for agentic PV debugging with three components: **PV-Bench**, 99 DRC and LVS debugging tasks across 5 real designs; **PV-World**, a reproducible GUI-based environment built entirely on the benchmark, open-source semiconductor technologies, and EDA tools; and **PV-Agent**, an expert-aligned multi-agent framework that grounds diagnosis and repair in design rules, layout geometry, and sign-off feedback. Experiments show that general-purpose agents struggle on PV-Bench in PV-World, with the strongest baseline solving only 23.2% of tasks. PV-Agent raises the pass rate to 54.5% on the same backbone and improves LVS debugging from 11.1% to 44.4%. PV-Debugger establishes PV as a challenging, automatically verifiable testbed for multimodal, tool-using agents and offers a foundation for developing specialized agents for real-world chip-design workflows.

1 INTRODUCTION

Modern electronic design automation (EDA) has automated much of the very large-scale integrated circuit (VLSI) design flow: Register-transfer-level (RTL) code can increasingly be generated automatically Liu et al. (2023); Ho et al. (2025), while placement and routing (P&R) tools of physical design transform circuit netlists synthesized from RTL code into chip layouts under engineer-specified constraints Kahng et al. (2011). Thus, an end-to-end implementation can be completed in roughly a day of compute (Olofsson, 2016). Yet, **a critical final stage remains largely manual** (Figure 1): *physical verification (PV)*. Performed immediately before sign-off, PV determines whether a layout is manufacturable and faithful to its intended circuit through DRC and layout-versus-schematic (LVS) comparison Saurabh (2023). Debugging their failures is challenging because verification tools identify *what is wrong*, but generally not *how to fix it*. Foundry rule decks specify how to detect DRC violations but provide no inverse procedure for constructing valid repairs. Moreover, P&R tools rely on simplified technology abstractions, leaving violations that emerge only under full sign-off rules or originate from macros and third-party intellectual property (IP). LVS is even more underdetermined: a mismatch indicates disagreement between layout and schematic without directly revealing its cause or repair. Consequently, engineers manually inspect reports, localize faults, diagnose geometric or connectivity causes, modify layouts, and rerun verification Li et al. (2006). This process is indispensable: a single escaped error can compromise fabrication, yet can consume weeks near tape-out and contribute substantially to late-stage schedule and engineering cost (Synopsys, Inc., 2022; Velivala, 2021).

Recently, large language model (LLM) and vision-language model (VLM) agents have shown strong potential for automating complex workflows, ranging from general-purpose computer interaction Agashe et al. (2025); Wang et al. (2026); Xie et al. (2024) to domain-specific VLSI tasks such as RTL generation Liu et al. (2023); Ho et al. (2025) and physical design Li et al. (2026); Ren et al. (2026); Wu et al. (2024); Liu et al. (2026b); Ghose et al. (2025). PV, however, remains largely unexplored. Existing studies focus primarily on synthesizing DRC checkers (Chang et al., 2025; Kim et al., 2026) or repairing local violations through programmatic layout manipulation (Ho et al.,

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.

2026; Mukherjee et al., 2026; Liu et al., 2026a). To our knowledge, no prior work systematically studies agentic LVS debugging or end-to-end sign-off debugging in the interactive environments expert engineers use.

PV is particularly well suited to agentic debugging for three reasons. First, it is *rule-referenced*: every violation is defined against an explicit rule or schematic that constrains valid repairs. Second, it is *experience-reusable*: recurring geometric and connectivity patterns allow successful strategies to be adapted across violations. Third, it is *verifiable*: rerunning DRC or LVS provides direct feedback on repair correctness. At the same time, PV also poses substantial challenges for current agents. It requires operating EDA *GUIs* to navigate reports, cross-probe errors, and modify layouts; performing *visual grounding* to associate textual violations with precise geometric structures; and executing *long-horizon debugging* involving repeated diagnosis, editing, verification, and recovery from unsuccessful repairs. PV therefore provides both an important automation problem and a challenging, automatically verifiable testbed for tool-using agents.

To address this problem systematically, we present **PV-Debugger**, a holistic framework for VLSI PV-Agentic debugging that integrates **a comprehensive benchmark (PV-Bench), an executable environment (PV-World), and an expert-aligned agentic workflow (PV-Agent)**. **PV-Bench** comprises 99 DRC and LVS debugging tasks from 5 real designs spanning two orders of magnitude in design size and multiple levels of complexity. **PV-World** is built entirely from an open-source process design kit (PDKs), designs, and EDA tools and integrates them into a standardized virtual environment, enabling reproducible evaluation without proprietary infrastructure. Evaluating state-of-the-art general-purpose agents with PV-Bench and PV-World reveals substantial limitations, particularly in EDA GUI interaction, visual grounding, violation diagnosis, and long-horizon repair. We therefore develop **PV-Agent**, a multi-agent framework that mirrors expert engineers' debugging workflow with joint GUI interaction, visual grounding, domain reasoning, long-horizon tool use, and automatic verification. Specifically, PV-Agent decomposes physical-verification debugging into specialized stages that inspect violations, reason about their causes, localize relevant layout structures, generate and execute repair operations, and verify the resulting design. This closed-loop workflow enables iterative debugging of both DRC violations and LVS mismatches and substantially improves debugging success over general-purpose agents.

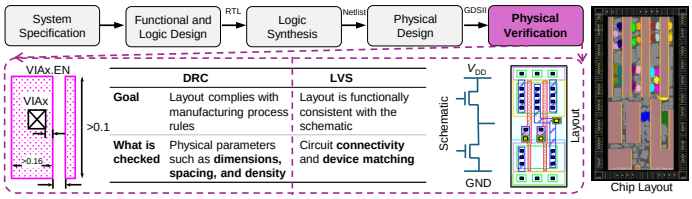


Figure 1: Modern VLSI design flow. Physical verification remains manual, one longstanding challenge in VLSI design.

Our key contributions are: **(1) PV-Bench**, a reproducible benchmark of 99 DRC and LVS debugging tasks across 5 real VLSI designs; **(2) PV-World**, a standardized, open-source execution environment for interactive PV debugging and sign-off-based evaluation; **(3) a systematic evaluation of state-of-the-art agents**, revealing key limitations in fault localization and long-horizon PV debugging; and **(4) PV-Agent**, an expert-aligned multi-agent framework that integrates diagnosis, localization, repair, and verification to substantially improve DRC and LVS debugging.

2 Background and Related Work

Physical Verification. PV is the final sign-off stage before delivering a VLSI layout for fabrication (Figure 1). After placement and routing (P&R), PV validates manufacturability and circuit consistency through two principal checks: DRC and LVS comparison. DRC checks layout geometry against foundry constraints such as spacing, width, enclosure, density, and antenna rules; LVS compares the circuit extracted from the layout against the reference schematic for device and connectivity mismatches. Although P&R tools perform intermediate DRC and repair, they rely on simplified technology abstractions for efficiency. Sign-off PV evaluates the completed layout against the full verification rule deck, exposing violations that remain after routing. Resolving them requires engineers to inspect reports, diagnose and localize faults, modify the layout, and rerun verification. As illustrated in Figure 2, DRC and LVS follow the same closed-loop workflow despite differing

in evidence and repair mechanisms. Importantly, verification reports identify violated constraints or mismatches but generally do not prescribe the corresponding repair. Moreover, repairs can introduce or expose additional violations, making PV debugging an inherently iterative, closed-loop process.

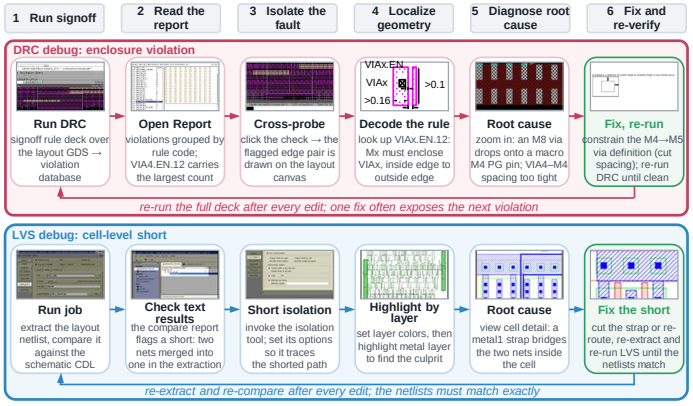


Figure 2: **DRC and LVS debugging in practice.** A DRC enclosure violation (top) and an LVS cell-level short (bottom) follow the same six-stage workflow: inspection, diagnosis, localization, repair, execution, and verification. Both require iterative debugging until sign-off checks pass.

Benchmarks for Physical Design. Recent benchmarks have enabled systematic evaluation of LLM- and VLM-based agents for physical design. ChatEDA-Bench Wu et al. (2024) and iScript-Bench Xu et al. (2026b) primarily evaluate the generation of P&R scripts, while PDAGENT-BENCH Li et al. (2026) covers a broader range of physical-design tasks. Rule2DRC Kim et al. (2026) focuses specifically on translating natural-language design rules into executable DRC checker code. These benchmarks therefore primarily evaluate *design-flow authoring*: generating scripts, commands, or verification programs that invoke EDA tools. PV debugging poses a fundamentally different problem. Given an observed sign-off failure, an agent must diagnose its cause, ground the reported error to physical layout structures, determine a valid repair, execute that repair, and verify its effect. Existing benchmarks do not evaluate this end-to-end debugging process and, in particular, do not cover LVS repair. **PV-Bench** fills this gap by evaluating agents on complete DRC and LVS debugging trajectories in an executable physical-verification environment, illustrated in Table. 2.

Agentic Frameworks for Physical Design. Existing approaches broadly fall into three categories. *Script-generation agents*, such as ChatEDA Wu et al. (2024) and iScript Xu et al. (2026b), translate design intent into P&R scripts. *Checker-generation agents*, including DRC-Coder Chang et al. (2025) and Rule2DRC Kim et al. (2026), synthesize executable DRC rules from textual specifications, learning how violations are *detected* rather than how observed violations should be *repaired*. *Closed-loop design agents*, including ORFS-Agent Ghose et al. (2025), OpenROAD Agent Wu et al. (2025), and PD-AGENT Li et al. (2026), iteratively interact with EDA tools to tune parameters, repair scripts, or optimize physical-design outcomes. SCALE Ho et al. (2026) comes closest to our setting by repairing local design-rule violations through self-supervised layout generation. These methods, however, predominantly generate scripts, commands, or structured design artifacts. Sign-off debugging instead requires reasoning from verification failures to their physical causes, localizing precise layout structures, executing repairs, and repeatedly validating their effects. LVS is particularly challenging because connectivity mismatches generally provide no direct geometric location or prescribed repair. Existing approaches therefore do not capture the complete interactive, long-horizon workflow of practical PV debugging.

3 Development of Benchmark and Operating Environment

Evaluating agents for PV requires more than a static dataset. A benchmark must be *faithful*, exercising the report interpretation, localization, tool interaction, and iterative repair required in practice; *reproducible*, ensuring identical initial design and tool states; and *fair*, providing a common execution and grading protocol. These requirements matter especially for PV because debugging is inherently interactive: a single repair may require multiple rounds of inspection, diagnosis, editing, and sign-off verification. We therefore introduce **PV-Bench**, a benchmark consisting of many tasks for end-to-end agentic debugging of DRC and LVS failures. We integrate benchmark tasks and their associated designs/gate netlists, along with an open-source implementation PDK and EDA tools,

Table 1: **PV-Bench.** 99 repair tasks over 5 open-source sky130 designs, counted per tier by the sign-off check that fails. *Easy*: exactly one DRC violation or one LVS mismatch. *Medium*: 4–8 injected copies of the same defect (median 6), sharing one root cause. *Hard*: 12–24 injected defects of several types (median 16), spanning multiple DRC rules and, for *Mixed* tasks, LVS as well. *Check* is one sign-off run. The lower panel lists the fault kinds covered, with their sky130 rule IDs.

			Easy		Medium		Hard		
Design	Instances	Check (s)	DRC	LVS	DRC	LVS	DRC	Mixed	Tasks
jpeg	108,462	517	3	4	5	5	1	2	20
aes	39,328	148	3	4	5	5	1	1	19
riscv32i	19,318	77	3	4	5	5	1	2	20
uart	1,767	7	3	4	5	5	1	2	20
gcd	919	4	3	4	5	5	1	2	20
Total			15	20	25	25	5	9	99
Tier total			35		50		14		

Check	Fault kinds covered
DRC	17 rules in 6 kinds: width (met3.1); spacing (met1.2, met2.2, met3.2, met4.2); minimum area (li.6, met3.6); via/contact enclosure (met1.4, met3.4, met4.3, via2.4a, via3.5); via spacing (via2.2, via3.2); device geometry (poly.4, poly.8, licon.14)
LVS	4 kinds: open (one net split in two), short (two nets merged), port mismatch, unmatched device

into a standardized virtual environment, i.e., **PV-World**, enabling reproducible evaluation without proprietary infrastructure. Agents operate within this environment for real-world debugging tasks.

3.1 PV-Bench: A Comprehensive Benchmark with Tiered Tasks

Task Construction and Difficulty. PV-Bench organizes tasks into three difficulty tiers that capture increasingly complex debugging scenarios. *Easy* tasks contain a single isolated violation, such as one LVS short or open, and test whether an agent can complete a full inspect-diagnose-localize-repair-verify cycle. *Medium* tasks contain multiple violations of the same type, requiring agents to identify a shared root cause and repair violations systematically rather than independently. *Hard* tasks combine multiple violation types with multiple instances, approximating practical sign-off debugging where repairs can interact and resolving one violation may expose or introduce another.

Benchmark Composition. PV-Bench contains **99 tasks across 5 designs** spanning two orders of magnitude in scale, from `gcd` with 919 instances to `jpeg` with 108K instances (Table 1). The tasks comprise 35 easy, 50 medium, and 14 hard tasks. Each easy or medium task fails a single sign-off check, giving 40 DRC tasks and 45 LVS tasks; of the 14 hard tasks, 5 span multiple DRC rules and 9 are *Mixed* tasks that fail both DRC and LVS. The tier mix is identical across designs, except that `aes` has one Mixed task rather than two. Medium tasks inject 4–8 copies of the same defect (median: 6), and hard tasks inject 12–24 defects of several types (median: 16). DRC faults cover 17 SKY130 rules in six categories: width, spacing, minimum area, via/contact enclosure, via spacing, and device geometry (poly and licon).

3.2 PV-World: Interactive Execution Environment

We develop **PV-World**, a standardized desktop virtual machine (VM) environment for executing PV-Bench tasks. Each task runs as an OSWorld Xie et al. (2024) task inside an Ubuntu virtual machine containing the SKY130A PDK Google and SkyWater Technology (2020), schematic (CDL), gate-level netlist, faulty GDSII layout, and a pinned sign-off toolchain: Magic 8.3.526 Ousterhout et al. (1985) for DRC and layout extraction and Netgen 1.5.293 Edwards (2026) for LVS (Appendix A.1). Agents observe 1920×1080 desktop screenshots and interact through mouse and keyboard with the terminal and Magic interface, allowing them to inspect reports, localize geometry, modify layouts, and rerun verification.

Sign-Off-Based Evaluation. PV-Bench evaluates the *resulting design*, not whether an agent reproduces a reference repair trajectory. After execution, the edited GDSII is independently re-verified using the pinned sign-off flow. A task succeeds only if it returns exactly to its task-specific baseline: every DRC rule must recover its baseline violation count and the LVS verdict must be restored. Task-specific baselines are necessary because real designs can contain pre-existing violations un-

Table 2: Comparison of PV-Bench with other LLM benchmarks for physical design.

Benchmark Features	ChatEDA-Bench (Wu et al., 2024)	DRC-Coder (Chang et al., 2025)	Rule2DRC (Kim et al., 2026)	SCALE (Ho et al., 2026)	**PV-Bench (Ours)**
Open-Source PDK	✓	×	✓	×	✓
Design Stage	Physical Design	Physical Design	Sign-off PV	Physical Design	Sign-off PV
Layout Scale	Block	Cell	Test layout	Local clip	Block
Multi-Step Task	–	×	×	×	✓
GUI Support	×	✓	×	✓	✓
LVS Debugging	×	×	×	×	✓
Ready-to-Run Environment[†]	×	×	×	×	✓
# of Cases	50	207	1,000	100	99
Target Tasks	P&R Flow Scripting	DRC Checker Code Generation	DRC Script Synthesis	DRC Violation Fixing	**DRC & LVS Debugging**

[†]**Ready-to-Run Environment**: ships the PDK, designs, and pinned EDA toolchain as a packaged image (PV-World), so every method runs in an identical setup with no per-method installation.

related to the debugging task. For example, `aes` ships with 104 residual DRC violations that no injected fault accounts for. Requiring zero violations would therefore make some tasks unsatisfiable, whereas merely reducing the violation count could reward destructive edits such as deleting geometry. Baseline restoration instead verifies that target faults are removed *without degrading the original design* while permitting multiple valid repair strategies.

Isolated and Integrity-Aware Grading. Grading is isolated from the agent to prevent evaluation manipulation. The edited layout is exported from PV-World and verified on the host in a network-disabled container with the pristine task bundle mounted read-only. The grader additionally flags layouts copied from the reference solution and layouts left unmodified. Thus, success requires a genuine repair validated independently by the underlying sign-off tools.

Overall, **PV-Bench** and **PV-World** provide three key capabilities: **(1) end-to-end debugging**, with 99 DRC and LVS tasks requiring diagnosis, localization, repair, and verification; **(2) reproducible interactive execution**, through a standardized EDA environment with pinned tools and identical initial states; and **(3) sign-off-grounded evaluation**, where independent DRC/LVS verification accepts multiple valid repairs while rejecting incomplete or destructive solutions.

4 PV-AGENT

PV-Bench shows that physical-verification debugging requires precise localization, domain-specific reasoning, and iterative verification beyond general-purpose GUI-use agents (see results in Section 5.2). We therefore introduce **PV-Agent**, an expert-aligned multi-agent framework that decomposes PV debugging into specialized roles and grounds repairs in executable sign-off feedback. Its central principle is: *an agent may propose a repair, but only the verifier can accept it*. Figure 3 illustrates the framework.

Expert-Aligned Multi-Agent Workflow. Rather than using a generic plan-act loop, PV-Agent mirrors the debugging procedure in Figure 2 through four specialized roles. The *Planner* converts DRC deltas and LVS mismatches into an ordered fault list. The *Locator* grounds each fault to relevant nets, layers, and coordinates. The *Fixer* converts this evidence into a minimal, replayable sequence of Magic operations. The *Reviewer* reruns DRC and LVS and compares the result against the task-specific baseline. Each role follows an explicit contract defining its tools, required evidence, outputs, and termination conditions. Roles communicate through a structured JSON journal that records diagnostic evidence, repair operations, and verification outcomes, making repairs replayable, revertible, and auditable. PV-Agent prioritizes connectivity before geometric repairs because reconnecting nets can introduce spacing, enclosure, or minimum-area violations; resolving these afterward avoids invalidating established connectivity. A fault is marked resolved only when confirmed by the sign-off checker.

Verifier-Grounded Repair. PV-Agent separates *proposing* a repair from *accepting* it. Rather than directly committing agent-generated edits, repair tools construct candidates from extracted geometry, execute them in a live Magic session, re-extract the layout, and rerun DRC and LVS. A candidate

is retained only if verification confirms the intended repair without unacceptable regressions; otherwise, it is reverted. For example, `route_open` and `cut_short` generate candidate operations for reconnecting opens and separating shorts, returning `accepted_ops` only after verification. This is critical because geometrically plausible repairs can be electrically incorrect. A direct connection between a pin and its target wire, for instance, may cross a VPWR or VGND rail, closing the open while creating a power short. Such errors may only emerge after extraction and LVS. PV-Agent therefore treats executable verification, rather than geometric plausibility, as the authority for repair correctness.

Layer-Aware Geometric Reasoning. PV faults are defined by both geometry and layer-specific connectivity. PV-Agent therefore resolves reported faults into explicit geometric primitives before repair. In SKY130, top-level labels do not necessarily correspond to hierarchical node names, while geometry directly beneath a label may belong to a nearby power rail. The Locator instead identifies the mask layers beneath a reported label, resolves the intended net on the corresponding routing layer, and extracts its geometry as exact rectangles. Repair dimensions are derived from the technology definition rather than assumed: contacts and vias differ in size and enclosure requirements, and minimum widths vary across metal layers. When a direct connection is blocked, PV-Agent can construct a higher-layer jumper using two vias and an intermediate wire with proper enclosure. It also explicitly converts among the coordinate systems used by verification reports, technology definitions, and Magic, enabling deterministic submicron repairs without screen-space measurement.

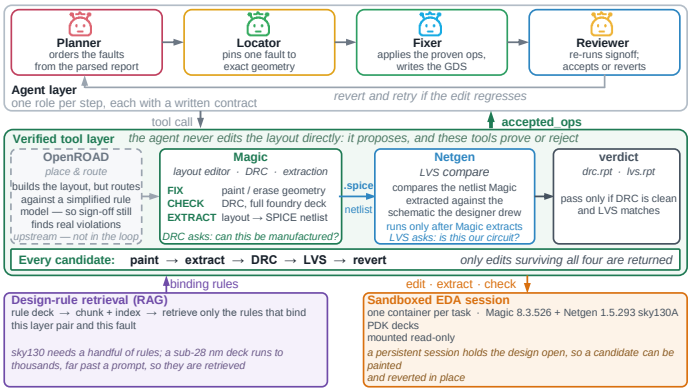


Figure 3: **PV-Agent.** Four roles pass one fault along the debugging loop of Figure 2, communicating only through a shared Jason journal. A verified tool layer settles every repair they consider by painting it into a live layout, re-extracting, checking DRC and LVS, and reverting; only candidates that survive all four are returned. Magic both edits and checks the layout, while LVS additionally requires Netgen, which cannot run until Magic has extracted.

Rule-Grounded Retrieval. Verification reports identify violations but rarely provide enough information to construct valid repairs. PV-Agent therefore retrieves relevant constraints from the PV rule corpus, including SKY130 rule summaries, executed DRC definitions, and Netgen configurations governing device matching and property tolerances. The corpus is indexed at rule-level granularity, allowing each query to retrieve only constraints relevant to the fault and its layer combination. Retrieved evidence supplies information absent from reports, including applicable layer sets, geometric constraints, and rule-specific exemptions. PV-Agent preserves the original rule text and identifier alongside generated explanations, providing traceability from each repair to its governing constraints. To prevent evaluation leakage, retrieval is restricted to rule sources and excludes benchmark tasks, reference solutions, and worked repair examples. This design also scales to technologies with rule decks too large to place directly in model context.

Command-Level Interaction Instead of Pixel-Level Search. Generic computer-use agents often search over action sequences and ground screenshot observations to pixel-level actions. PV-Agent replaces both with domain-native structure where available. The high-level debugging workflow is known (Figure 2); the challenge is diagnosing the fault and constructing a valid repair, not discovering an arbitrary sequence of actions. Thus, PV-Agent encodes the workflow explicitly and focuses model reasoning on localization and repair. Moreover, EDA operations such as loading layouts, resolving nets, extracting rectangles, editing geometry, and invoking verification admit precise Magic or shell commands. PV-Agent reasons over extracted geometric representations and executes deterministic commands rather than estimating repair coordinates from pixels. Screenshots remain

available for inspection and ambiguous cases, but geometric measurements come directly from the layout database. By replacing action search and pixel-level grounding with an expert workflow and exact geometric operations, PV-Agent focuses model capacity on the semantic challenges of PV: diagnosing root causes, selecting applicable rules, and constructing repairs that survive sign-off.

5 Experiments

5.1 Experimental Setup

Models and Baselines. We evaluate five backbone models: four proprietary models (GPT-5.6-sol and GPT-5.6-luna OpenAI (2026), Claude Sonnet 5 Anthropic (2026c), and Claude Opus 5 Anthropic (2026b)) and the open-weight DeepSeek-V4.1-Flash Xu et al. (2026a). As general-purpose computer-use baselines, we evaluate Agent S3 Gonzalez-Pumariega et al. (2025), OpenAI's Computer-Using Agent (CUA) OpenAI (2025), and Anthropic's computer-use agent Anthropic (2026a). All experiments run on a server with eight NVIDIA A6000 Ada GPUs and an AMD EPYC 7313 CPU. **EDA Environment.** All designs target the open-source SkyWater SKY130A process Google and SkyWater Technology (2020) and are implemented with Yosys Wolf et al. (2013) and OpenROAD Ajayi et al. (2019). Magic 8.3.526 Ousterhout et al. (1985) performs DRC, layout extraction, and layout editing, while Netgen 1.5.293 Edwards (2026) performs LVS; Appendix A.1 describes each tool. All agents operate on the same PV-World task instances and are evaluated using the external sign-off procedure described in Section 3.2. **Metrics.** Our primary metric is *task pass rate*. A task passes only when the repaired layout returns exactly to its task-specific sign-off baseline: the violation count for every DRC rule must match the baseline, and the LVS verdict must be restored. We additionally report token usage, model calls, screenshots, and wall-clock runtime to quantify debugging cost.

5.2 Evaluation Results on PV-Bench

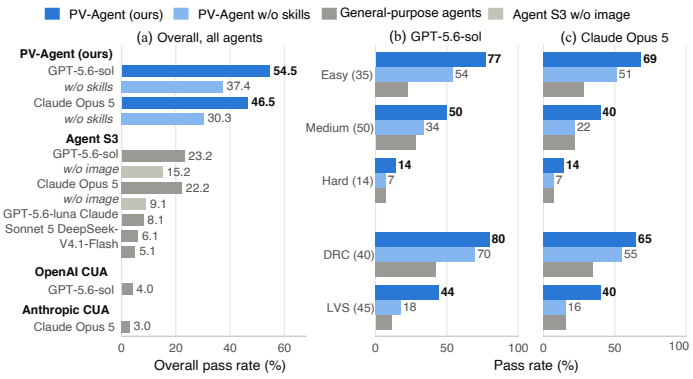


Figure 4: **Performance on PV-Bench.** (a) Overall pass rate across agents and backbones. (b,c) Pass rate by difficulty tier and fault type for PV-Agent, its ablation without skills, and Agent S3 using the same backbone. Hard tasks, which span multiple DRC rules or mix DRC and LVS faults, are reported separately. Exact results are provided in Table 3.

Table 3 and Figure 4 summarize the main results.

General-purpose Agents Struggle with PV debugging. The strongest general-purpose baseline, Agent S3 with GPT-5.6-sol, solves only 23.2% of the 99 evaluated tasks. With GPT-5.6-luna, Claude Sonnet 5, or DeepSeek-V4.1-Flash, its pass rate drops to 5.1–8.1%. OpenAI CUA and Anthropic's computer-use agent achieve only 4.0% and 3.0%, respectively, despite using the same backbones as the two strongest Agent S3 configurations. Moreover, every task solved by these two computer-use agents is an easy DRC case; neither solves a medium, hard, or LVS task. *These results indicate that general-purpose computer-use capabilities alone are insufficient for realistic PV debugging.*

LVS Exposes a Particularly Large Capability Gap. All general-purpose baselines perform substantially worse on LVS than on DRC. Agent S3 with GPT-5.6-sol solves 42.5% of DRC tasks but only 11.1% of LVS tasks. The strongest baseline LVS result is 15.6% with Claude Opus 5; apart from these two Agent S3 configurations, the remaining baselines solve at most one of the 45 LVS tasks. This gap reflects a fundamental difference in localization: DRC reports identify a violated rule and its geometric region, whereas LVS reports identify mismatched nets or devices without directly

Table 3: **Pass rate (%) on PV-Bench.** A task passes when its repaired layout matches the design's sign-off baseline in both DRC and LVS. Parentheses give the number of tasks per column. *DRC* and *LVS* split the easy and medium tasks by the kind of injected fault; the 14 hard tasks, 5 spanning multiple DRC rules and 9 mixing DRC and LVS faults, appear only under *Hard*. Best in **bold**, second best underlined. With the same backbone, PV-Agent raises the overall pass rate of Agent S3 from 23.2 to 54.5 (GPT-5.6-sol) and from 22.2 to 46.5 (Claude Opus 5).

			By tier			By fault kind	
Agent	Backbone	Overall (99)	Easy (35)	Medium (50)	Hard (14)	DRC (40)	LVS (45)
Agent S3	GPT-5.6-sol	23.2	22.9	28.0	7.1	42.5	11.1
	Claude Opus 5	22.2	28.6	22.0	7.1	35.0	15.6
	GPT-5.6-luna	8.1	20.0	2.0	0.0	17.5	2.2
	Claude Sonnet 5	6.1	17.1	0.0	0.0	15.0	0.0
	DeepSeek-V4.1-Flash	5.1	14.3	0.0	0.0	12.5	0.0
OpenAI CUA	GPT-5.6-sol	4.0	11.4	0.0	0.0	10.0	0.0
Anthropic CUA	Claude Opus 5	3.0	8.6	0.0	0.0	7.5	0.0
PV-Agent (ours)	GPT-5.6-sol	**54.5**	**77.1**	**50.0**	**14.3**	**80.0**	**44.4**
	Claude Opus 5	46.5	68.6	40.0	**14.3**	65.0	40.0

locating the responsible geometry. Successful LVS repair therefore requires tracing connectivity through the extracted layout before a repair can even be formulated.

PV-Agent Substantially Improves both DRC and LVS Debugging. Using identical backbones, PV-Agent increases overall pass rate from 23.2% to 54.5% with GPT-5.6-sol and from 22.2% to 46.5% with Claude Opus 5, corresponding to 2.3× and 2.1× improvements over Agent S3. PV-Agent also outperforms Agent S3 across every difficulty tier and fault category (Figures 4b,c). The improvement is particularly pronounced on LVS. With GPT-5.6-sol, PV-Agent reaches 44.4% LVS pass rate compared with 11.1% for Agent S3; with Claude Opus 5, the corresponding rates are 40.0% and 15.6%. DRC performance reaches 80.0% with GPT-5.6-sol. These results suggest that explicit localization, rule-grounded reasoning, and verifier-grounded repair are especially valuable when the verification report does not directly reveal where the underlying fault resides. Nevertheless, complex debugging remains challenging. PV-Agent solves only 2 of the 14 hard tasks with either backbone, leaving substantial headroom for reasoning over interacting violations and longer repair trajectories.

5.3 Ablation Study

Domain-Specific Skills are Critical, Especially for LVS. Removing PV-Agent's specialized skills reduces overall pass rate from 54.5% to 37.4% with GPT-5.6-sol and from 46.5% to 30.3% with Claude Opus 5. The degradation is substantially larger for LVS: pass rate falls from 44.4% to 17.8% with GPT-5.6-sol and from 40.0% to 15.6% with Claude Opus 5. By comparison, DRC decreases from 80.0% to 70.0% and from 65.0% to 55.0%, respectively. This asymmetry reinforces that LVS depends more strongly on domain-specific localization and connectivity reasoning.

General-Purpose Agents Depend Heavily on Visual Input. Removing image input from Agent S3 decreases its overall pass rate from 23.2% to 15.2% with GPT-5.6-sol and from 22.2% to 9.1% with Claude Opus 5. In contrast, PV-Agent obtains geometric information directly from the layout database and uses screenshots only selectively. This result highlights a key architectural difference: general-purpose GUI agents must infer state primarily from rendered pixels, whereas PV-Agent exploits exact domain-native representations when available.

5.4 Cost analysis

Table 4 reports the computational cost of PV-Agent over the 99 evaluated tasks. PV-Agent requires an average of 65.8 model calls per task with GPT-5.6-sol and 85.6 with Claude Opus 5, corresponding to 191K and 250K tokens, respectively. Approximately 99% of these tokens are input tokens, indicating that cost is dominated by contextual reasoning rather than generated text. GPT-5.6-sol is more efficient in our experiments: it achieves a higher pass rate (54.5% vs. 46.5%) while consuming 24% fewer tokens, corresponding to approximately 350K tokens per solved task compared with 537K for Claude Opus 5. End-to-end runtime averages 19.2 minutes per task with GPT-5.6-sol and

Table 4: **Cost of PV-Agent on PV-Bench.** Total, prompt, and completion tokens are summed over all 99 tasks; per-task values divide by 99, and per-solved-task values by the number of tasks passed. *Model* is the time spent in model API calls; *Wall* is the wall-clock time of the whole run.

		Tokens					Per task			
Backbone	Pass (%)	Total (M)	Prompt (M)	Compl. (K)	Per task (K)	Per solved task (K)	API calls	Model (s)	Wall (min)	Screen-shots
GPT-5.6-sol	54.5	189.1	18.67	202	191	350	65.8	398	19.2	7
Claude Opus 5	46.5	247.5	24.40	301	250	537	85.6	440	15.6	12.9

15.6 minutes with Claude Opus 5. Model inference accounts for only 35% and 47% of this runtime, respectively; the remainder is dominated by EDA tool execution. Verification itself can be expensive, i.e., a single sign-off run takes up to 517 s on `jpeg` (Table 1), making unnecessary verification attempts costly. PV-Agent also requires relatively few visual observations: 7.0 screenshots per task with GPT-5.6-sol and 12.9 with Claude Opus 5. Rather than repeatedly inferring geometry from screenshots, PV-Agent queries exact geometric information from the layout database and reserves visual observations for inspection and ambiguity resolution.

5.5 Failure Analysis and Open Challenges

Localization is the Dominant Failure Mode. The primary bottleneck is identifying *where* to repair rather than executing the repair itself. Across failed runs, 76–84% terminate without modifying the layout (Appendix A.6). In contrast, once PV-Agent commits an edit, 83% of those edits ultimately pass sign-off. This gap is particularly visible for LVS: with GPT-5.6-sol, PV-Agent solves 80.0% of DRC tasks but only 44.4% of LVS tasks. These results identify fault localization, especially tracing LVS mismatches back to physical geometry, as a central challenge for future agentic PV systems.

Long-Horizon, Interacting Faults Remain Difficult. Even the strongest PV-Agent configuration solves only 54.5% of all tasks and 2 of 14 hard tasks. Hard cases require reasoning over multiple interacting faults, where one repair may expose or introduce another violation. Improving state tracking, credit assignment across repairs, and reasoning over interacting geometric and connectivity constraints therefore remains an important direction.

Efficiency and Confidentiality Motivate Specialized Local Models. Current frontier-model configurations consume 191K–250K tokens and 16–19 minutes per task. Meanwhile, the image ablation shows that generic GUI agents cannot simply discard visual observations without substantial performance loss. In practical semiconductor workflows, proprietary layouts and foundry technology information may also be unsuitable for external model APIs. These considerations motivate smaller domain-specialized models that can operate locally. PV-World provides a potential substrate for developing such models because sign-off verification supplies an automatically computed success signal for generated repairs.

Advanced Technology Nodes Present a Aubstantially Harder Regime. PV-Bench targets SKY130, which is comparatively simple relative to modern commercial technologies. Advanced nodes contain more routing layers and substantially larger, more context-dependent rule sets, increasing both the search space for localization and the constraints governing valid repairs. Extending agentic PV debugging to these environments will require stronger geometric and connectivity reasoning, scalable rule retrieval, and efficient interaction with increasingly expensive sign-off tools.

6 Conclusion

We presented **PV-Debugger**, a framework for agentic debugging of physical verification built entirely on open-source PDKs, designs, and tools. It comprises **PV-Bench**, to our knowledge the first benchmark for end-to-end DRC and LVS debugging on real designs; **PV-World**, a reproducible environment; and **PV-Agent**, a multi-agent system tailored to DRC and LVS debugging. Benchmarking state-of-the-art computer-use agents on PV-Bench shows that they rarely complete PV debugging. PV-Agent reaches 54.5%, more than twice the strongest baseline on the same backbone, although LVS and hard multi-fault tasks remain far from solved. We hope PV-Debugger lays the foundation for agentic PV debugging and, ultimately, for fully automated physical verification.

AI USE STATEMENT

We used large language models (LLMs) for writing assistance. LLMs were used to polish the wording of the paper and to check consistency between the text and tables. The research ideas, benchmark, method design, experiments, and analysis were developed by the authors, who take full responsibility for all content of this paper.

REPRODUCIBILITY STATEMENT

We describe every component needed to reproduce our results. PV-Bench's designs, fault injection, and composition are given in Table 1 and Appendices A.2 and A.3, and its sign-off grading protocol, with task-specific baselines and integrity checks, in Section 3.2. PV-World is built only from open-source components with pinned versions: the SKY130A PDK, Magic 8.3.526, and Netgen 1.5.293 (Appendix A.1). PV-Agent's roles, tools, and skills are described in Section 4 and Appendix A.4. We will release all data upon acceptance, including the PV-Bench tasks, the PV-World environment, the sign-off grader, and the agent trajectories from our experiments.

REFERENCES

Saaket Agashe, Jiuzhou Han, Shuyu Gan, Jiachen Yang, Ang Li, and Xin Wang. Agent s: An open agentic framework that uses computers like a human. In *International Conference on Learning Representations*, volume 2025, pp. 22924–22946, 2025.

Tutu Ajayi, Vidya A Chhabria, Mateus Fogaça, Soheil Hashemi, Abdelrahman Hosny, Andrew B Kahng, Minsoo Kim, Jeongsup Lee, Uday Mallappa, Marina Neseem, et al. Toward an open-source digital flow: First learnings from the openroad project. In *Proceedings of the 56th Annual Design Automation Conference 2019*, pp. 1–4, 2019.

Anthropic. Computer Use Tool. Claude API documentation, 2026a. URL https://platform.claude.com/docs/en/agents-and-tools/tool-use/computer-use-tool. Accessed: 2026-09-25.

Anthropic. Claude Opus 5 system card, 2026b. URL https://www.anthropic.com/claude-opus-5-system-card. Accessed: 2026-09-26.

Anthropic. Claude Sonnet 5 system card, 2026c. URL https://www.anthropic.com/claude-sonnet-5-system-card. Accessed: 2026-09-26.

Chen-Chia Chang, Chia-Tung Ho, Yaguang Li, Yiran Chen, and Haoxing Ren. Drc-coder: Automated drc checker code generation using llm autonomous agent. In *Proceedings of the 2025 International Symposium on Physical Design*, pp. 143–151, 2025.

R. Timothy Edwards. Netgen: LVS and netlist comparison tool. Open Circuit Design, 2026. URL https://opencircuitdesign.com/netgen/. Accessed: 2026-09-26.

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. IEEE, 2025.

Gonzalo Gonzalez-Pumariega, Vincent Tu, Chih-Lun Lee, Jiachen Yang, Ang Li, and Xin Eric Wang. Scaling agents for computer use. *arXiv preprint arXiv:2510.02250*, 2025.

Google and SkyWater Technology. SkyWater SKY130 PDK, 2020. URL https://skywater-pdk.readthedocs.io/en/main/. Accessed: 2026-09-26.

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.

Chia-Tung Ho, Haoyu Yang, Guanglei Zhou, Yoshi Nishi, Yaguang Li, Walker Turner, Cunxi Yu, Yiran Chen, and Brucek Khailany. Scale: Self-supervised constraint-aware layout generation for local p&r drv fixing at advanced nodes. *arXiv preprint arXiv:2607.21850*, 2026.

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

Jinuk Kim, Junsoo Byun, Donghwi Hwang, Seong-Jin Park, and Hyun Oh Song. Rule2drc: Benchmarking llm agents for drc script synthesis with execution-guided test generation. *arXiv preprint arXiv:2605.15669*, 2026.

Qiufeng Li, Rongqian Chen, Quan Cheng, Chengxuan Wang, Sizhe Tang, Chia-Tung Ho, David Z Pan, Tian Lan, and Weidong Cao. Pdagent-bench: Characterizing, grounding, and architecting llm/vlm agents for vlsi physical design. *arXiv preprint arXiv:2606.17253*, 2026.

Wei Li, Daniel Blakely, Scott Van Sooy, Keven Dunn, David Kidd, Robert Rogenmoser, and Dian Zhou. Lvs verification across multiple power domains for a quad-core microprocessor. *ACM Transactions on Design Automation of Electronic Systems (TODAES)*, 11(2):490–500, 2006.

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.

Pengju Liu, Nuo Xu, Jinwei Tang, Yu Cao, and Caiwen Ding. Bridging the last mile of circuit design: Posteda-bench, a hierarchical benchmark for ppa convergence and drc fixing. *arXiv preprint arXiv:2605.06936*, 2026a.

Yang Liu, Shiwei Hou, Xiyuan Chen, Yu Wang, Sen Yuan, Qirui Gan, Shao You, Feifan Chen, Wencheng Li, Shuyang Hu, et al. Zhulong: Execution-grounded llm agent for eda scripting with offline api self-exploration. *arXiv preprint arXiv:2608.07925*, 2026b.

Anushka Mukherjee, Kang He, and Kaushik Roy. Drc-aid: Design-rule correction via agentic framework utilizing inference-time large language models. *arXiv preprint arXiv:2607.22761*, 2026.

Andreas Olofsson. Epiphany-v: A 1024 processor 64-bit risc system-on-chip. *arXiv preprint arXiv:1610.01832*, 2016.

OpenAI. Computer-Using Agent, January 2025. URL `https://openai.com/index/computer-using-agent/`. Accessed: 2026-09-25.

OpenAI. GPT-5.6 system card, 2026. URL `https://deploymentsafety.openai.com/gpt-5-6`. Accessed: 2026-09-26.

John K Ousterhout, Gordon T Hamachi, Robert N Mayo, Walter S Scott, and George S Taylor. The magic vlsi layout system. *IEEE Design & Test of Computers*, 2(1):19–30, 1985.

Shuo Ren, Yaohui Han, Libo Shen, Zhiqiang Jia, Rongliang Fu, Bei Yu, and Tsung-Yi Ho. Agenticeco: An agentic framework for eco on 3d integrated circuits. *arXiv preprint arXiv:2608.03738*, 2026.

Sneh Saurabh. *Introduction to VLSI design flow*. Cambridge University Press, 2023.

Synopsys, Inc. Faster and smarter LVS for the SoC era. White paper, 2022.

Srinivas Velivala. MaxLinear and Calibre RealTime Digital: Signoff DRC in P&R. Siemens EDA case study, 2021.

Xinyuan Wang, Bowen Wang, Dunjie Lu, Junlin Yang, Tianbao Xie, Junli Wang, Jiaqi Deng, Xiaole Guo, Yiheng Xu, Chen Wu, et al. Opencua: Open foundations for computer-use agents. *Advances in Neural Information Processing Systems*, 38:139756–139806, 2026.

Clifford Wolf, Johann Glaser, and Johannes Kepler. Yosys-a free verilog synthesis suite. In *Proceedings of the 21st Austrian Workshop on Microelectronics (Austrochip)*, volume 97, pp. 1–6, 2013.

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. IEEE, 2025.

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.

Tianbao Xie, Danyang Zhang, Jixuan Chen, Xiaochuan Li, Siheng Zhao, Ruisheng Cao, Toh J Hua, Zhoujun Cheng, Dongchan Shin, Fangyu Lei, et al. Osworld: Benchmarking multimodal agents for open-ended tasks in real computer environments. *Advances in Neural Information Processing Systems*, 37:52040–52094, 2024.

Anyi Xu, B Li, Bangcai Lin, Bing Xue, BingCheng Xian, Bingzheng Xu, Bochao Wu, Bowei Zhang, Boyi Deng, CC Yu, et al. Deepseek-v4. 1-flash: Pushing the limits of kv cache compression. *arXiv preprint arXiv:2609.19969*, 2026a.

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*, 2026b.

A Appendix

A.1 EDA Tools and PDKs

PV-World is built entirely from open-source components, which is what allows the PDK, the designs, and the pinned toolchain to be shipped together as a single ready-to-run image. Commercial sign-off tools such as Calibre, IC Validator, and PVS run complete foundry decks for advanced nodes, but they cannot be redistributed in a public image, so a benchmark built on them could not guarantee that every method runs in an identical setup.

SkyWater 130 nm. SKY130 (Google and SkyWater Technology, 2020) is an open-source PDK for SkyWater Technology's 130 nm process, released in collaboration with Google. Unlike predictive academic PDKs it is manufacturable, and designs built on it have been taped out repeatedly through open shuttle programs. It ships complete design-rule and device models, so physical verification (DRC and LVS) against it reflects the rules of a real foundry process. PV-Bench uses the `sky130A` variant of the PDK with the high-density standard-cell library (`sky130_fd_sc_hd`).

Yosys and OpenROAD. The layouts in PV-Bench come from an open-source RTL-to-GDSII flow. Yosys (Wolf et al., 2013) synthesizes each design's RTL into a gate-level netlist of SKY130 standard cells, and OpenROAD (Ajayi et al., 2019) takes that netlist through floorplanning, placement, clock-tree synthesis, and routing to produce the final GDSII layout; both are driven by OpenROAD-flow-scripts (ORFS) on the `sky130hd` platform. The router checks design rules during routing, but against a simplified rule model rather than the full foundry deck, so its output must still pass sign-off DRC and LVS; this is also why real designs can retain residual violations that the task-specific baselines of Section 3.2 account for. PV-World ships the same flow so that agents can invoke broader physical-design operations when needed. It is appended to, rather than prepended to, `PATH`, so sign-off commands always resolve to the pinned Magic and Netgen versions used for grading.

Magic. Magic (Ousterhout et al., 1985) is a VLSI layout editor originally developed at UC Berkeley and now maintained by Open Circuit Design. Beyond editing, it reads and writes GDSII, checks DRC against the rules in the technology file, and extracts a circuit netlist from the layout geometry. In PV-World, Magic 8.3.526 plays three roles: it is the DRC checker, the extractor that produces the layout-side netlist for LVS, and the editor in which agents inspect and repair the layout. Agents can operate it through its graphical layout window or through its Tcl console, where commands such as `box`, `paint`, `erase`, `drc`, and `extract` act on exact coordinates. General-purpose GUI agents mostly work through the window, whereas PV-Agent issues these commands directly and decides repairs from rectangle dumps rather than from pixels (Section 4).

Netgen. Netgen (Edwards, 2026), also maintained by Open Circuit Design, is an LVS tool that compares two netlists, in SPICE or Verilog format, and reports mismatched nets, devices, pins, and device properties. How devices are matched is governed by a technology-specific setup file, which specifies, for example, which device terminals are permutable, whether parallel or series devices are merged, and the tolerance on each device property. In PV-World, Netgen 1.5.293 compares the netlist that Magic extracts from the layout against the design's reference netlist, using the SKY130A

setup file. LVS therefore cannot run until Magic has re-extracted the edited layout, and its report names mismatched nets and devices rather than layout coordinates, which is why LVS repair requires tracing connectivity back to geometry (Section 5.2). Magic and Netgen together form the DRC and LVS sign-off pair of the open-source OpenLane flow used for SKY130 tape-outs, and PV-World pins both so that the checks agents run inside the VM are identical to those the external grader runs.

A.2 PV-BENCH DESIGNS

Table 5: **PV-Bench designs**, all from the OpenROAD-flow-scripts design suite and implemented on the SKY130 `sky130hd` platform. *Base. DRC* is the number of residual violations the signed-off layout already carries, which every task on that design must return to; all five designs match LVS at baseline.

Design	Top module	Description	Instances	Base. DRC
`gcd`	`gcd`	Greatest-common-divisor unit generated with PyMTL; ORFS uses it to sanity-check the flow	919	0
`uart`	`uart`	AXI4-Stream UART transmitter and receiver by Alex Forencich	1,767	28
`riscv32i`	`riscv`	Simple RV32I processor core written for an undergraduate course	19,318	14
`aes`	`aes_cipher_top`	AES-128 encryption core from OpenCores	39,328	104
`jpeg`	`jpeg_encoder`	JPEG encoder from the OpenCores Video Compression Systems project	108,462	92

PV-Bench uses five designs from the OpenROAD-flow-scripts (ORFS) design suite (Ajayi et al., 2019), listed in Table 5. They cover different kinds of logic, from a small arithmetic unit and a serial interface to a processor core, a cipher, and an image encoder, and span more than two orders of magnitude in size. ORFS provides reference `sky130hd` configurations for four of them; the reference configuration of `uart` targets ASAP7, so it is ported to `sky130hd`. Each design is taken through the flow to a signed-off GDSII layout, from which the faulty task layouts are derived. Four of the five layouts already carry residual DRC violations from the router and the standard cells. These belong to no task, which is why grading compares each task against its design's own baseline rather than against zero (Section 3.2).

A.3 FAULT INJECTION

Every PV-Bench task starts from the signed-off layout of one of the five designs, into which we manually inject DRC violations, LVS mismatches, or both. Each fault is a small, deliberate edit of the layout geometry rather than an error written into a report: a DRC fault moves, resizes, or reshapes a shape until it violates one sky130 rule; an LVS open removes part of a net's wiring and splits the net in two; an LVS short adds metal that merges two nets; and port and device faults disturb a pin or a device so that Netgen can no longer match it. The faults are deliberately diverse. DRC faults cover 17 rules in six categories on the local-interconnect, metal1–metal4, via2, via3, poly, and licon layers, and LVS faults cover four kinds of mismatch (Table 1). They are spread across all five designs and all three difficulty tiers, and medium and hard tasks combine 4–8 and 12–24 injected defects, respectively, so that repairs share a root cause or interact. Task prompts state only the goal, returning sign-off to the baseline, and give no fault class, rule, count, or location; task identifiers are anonymized, so every fault must be diagnosed from the sign-off reports.

A.4 PV-AGENT SKILLS

A *skill* is the procedural knowledge given to one PV-Agent role, supplied as that role's system prompt together with the tool scripts it invokes. PV-Agent has five skills, one each for the Planner, Locator, Fixer, and Reviewer, and a rule-lookup skill consulted before any edit. They combine three kinds of content. (1) *Tool scripts* operate Magic and Netgen reliably: resolving a net from a report label (`net_info`), measuring the gap between two nets and what lies between them (`bridge_gap`, `corridor`), trying candidate repairs in the live layout and returning only the accepted ones (`route_open`, `cut_short`, `spacing_candidates`), checking DRC on a local

box, and reverting the layout from disk. (2) *Worked repair cases* show how DRC spacing violations and LVS opens and shorts are taken from the report to an accepted repair. (3) *Experience* distilled from earlier runs records the pitfalls that repeatedly broke repairs, such as a bare `select net` under a label selecting the ground rail, a straight bridge shorting a power rail, or `undo` failing after an erase, together with working rules such as repairing connectivity before geometry and writing the layout once per fault. The *w/o skills* ablation removes all five skills, including their tool scripts; what remains is the four-role workflow with its journal, the sign-off checker, and direct shell and Magic Tcl access.

Generality. The skills are written to be design-agnostic. They describe how to operate the tools and how to debug each class of fault, contain no per-design logic or parameters, and are used unchanged for every design, fault, and backbone. Their tools act on whatever geometry the checker reports and derive every repair dimension from the technology file, so they apply to any sky130 layout. At run time, agents are sandboxed from the reference solutions, the worked repairs shipped with the benchmark, and the grader's internals, and retrieval is limited to rule sources (Section 4).

A.5 Terminal-Agent Baseline and Screenshot Ablation

Table 6: **Screenshot ablation** (pass rate, %). *w/o screenshots* disables PV-Agent's screenshot tool; Agent S3 without images acts only through the shell and Magic's Tcl interface and serves as the terminal-agent baseline. Parentheses give the number of tasks per column.

			By tier			By fault kind	
Backbone	Agent	Overall (99)	Easy (35)	Medium (50)	Hard (14)	DRC (40)	LVS (45)
GPT-5.6-sol	PV-Agent	54.5	77.1	50.0	14.3	80.0	44.4
	PV-Agent w/o screenshots	42.4	62.9	38.0	7.1	60.0	37.8
	Agent S3 w/o images	15.2	–	–	–	–	–
Claude Opus 5	PV-Agent	46.5	68.6	40.0	14.3	65.0	40.0
	PV-Agent w/o screenshots	37.4	68.6	24.0	7.1	55.0	31.1
	Agent S3 w/o images	9.1	–	–	–	–	–

Agent S3 as a terminal agent. A natural baseline for PV debugging is a general coding or terminal agent with shell and Magic Tcl access. Agent S3 already contains one: besides GUI actions, its code agent executes shell commands, and in PV-World it controls Magic through Magic's Tcl interface. With image input removed, Agent S3 observes the environment only through command output and acts only through the shell and Magic Tcl, so it serves as this baseline. It solves 15.2% of the tasks with GPT-5.6-sol and 9.1% with Claude Opus 5 (Table 6).

PV-Agent without screenshots. Disabling PV-Agent's screenshot tool lowers its pass rate from 54.5% to 42.4% with GPT-5.6-sol and from 46.5% to 37.4% with Claude Opus 5. Although PV-Agent takes only 7.0 and 12.9 screenshots per task (Table 4), these observations still contribute, most visibly on DRC with GPT-5.6-sol, which falls from 80.0% to 60.0%. Without any visual input, however, PV-Agent still solves 2.8× and 4.1× as many tasks as the terminal-agent baseline on the same backbones. Its advantage over a terminal agent therefore comes from its workflow, tools, and skills rather than from visual input.

A.6 Failure analysis

Table 7 classifies the outcome of every task by whether the agent changed the layout and, if it did, which sign-off check still fails.

Most failures never reach an edit. For both frameworks, the dominant failure is leaving the layout unchanged. Agent S3 makes no edit on 63 (GPT-5.6-sol) and 65 (Claude Opus 5) of the 99 tasks, which is 83% and 84% of its failures: it spends its steps running the checker, reading reports, and inspecting geometry without ever committing a change, so it rarely gets as far as a repair. PV-Agent with GPT-5.6-sol leaves the layout unchanged on 34 tasks, 76% of its failures. Because PV-Agent reverts every candidate edit that fails re-verification (Figure 3), most tasks it cannot solve end with the original layout rather than a damaged one.

Table 7: **Outcome of every PV-Bench task** (counts out of 99). *No edit*: the final layout is byte-identical to the input; the agent ran the checker, read reports, and inspected geometry but never wrote a change. *Edited, still failing*: the agent changed the layout, but the final sign-off still fails on DRC, LVS, or both. *Layout destroyed*: the output GDS is missing or empty, typically after a Magic `gds write` with no technology loaded. No run ended in a harness error or any other failure mode.

			Failed				
			No	Edited, still failing			Layout
Agent	Backbone	Passed	edit	DRC	LVS	Both	destroyed
Agent S3	GPT-5.6-sol	23	63	9	4	0	0
	Claude Opus 5	22	65	8	3	1	0
PV-Agent (ours)	GPT-5.6-sol	54	34	5	4	2	0

PV-Agent attempts more repairs, and more of them succeed. PV-Agent edits the layout on 65 tasks, compared with 36 and 34 for Agent S3, and 83% of its edits pass sign-off, compared with 64% and 65%. Its 11 edited failures are spread across DRC (5), LVS (4), and both (2), whereas most of Agent S3's edited failures fail on DRC alone (9 of 13 and 8 of 12). No run ended in a harness error, and no agent destroyed a layout.

A.7 Case Studies

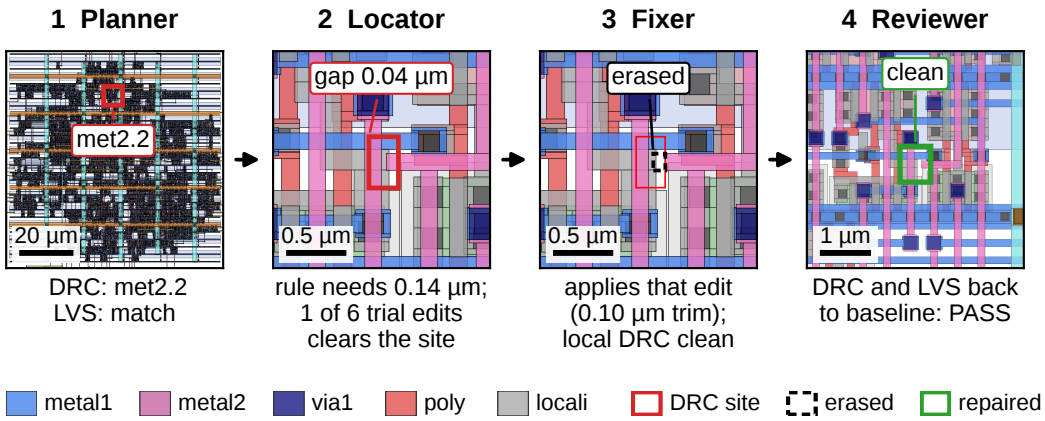


Figure 5: **Case study: a DRC repair that passes** (task pv-0013, `gcd`, easy; gpt-5.6-sol backbone). (1) Sign-off reports met2.2 (metal2 spacing) violation at one site. (2) The Locator finds a 0.04 μm gap where 0.14 μm is required and tests six candidate edits in the live layout. (3) The Fixer applies the only edit that clears the site, a 0.10 μm trim of the wire end. (4) Full sign-off returns DRC and LVS to the baseline.

Figures 5–6 show two easy tasks for PV-Agent on `gcd`: one it solves, and one it fails both with its own tools and when operating through the desktop GUI. All three runs use the gpt-5.6-sol backbone.

Success: a DRC spacing repair. Sign-off reports met2.2 violation at one site and a matching LVS verdict, so the Planner schedules a single DRC fault. The DRC report already gives the site's coordinates, so the Locator only has to read the geometry there: the end of a horizontal metal2 wire sits 0.04 μm from a vertical metal2 wire, whereas met2.2 requires 0.14 μm. Its spacing tool then tries six candidate edits in the live layout, each moving one edge of a nearby metal2 shape; only one clears the site while keeping the minimum wire width, a 0.10 μm trim of the wire end. The Fixer applies exactly that edit and confirms a clean local DRC, and the Reviewer's full sign-off returns DRC and LVS to the baseline. The whole run takes 8 model calls, 53K tokens, and 59 s.

Failure: an LVS open that is never localized. LVS reports that the output of buffer `buf_1_36` is disconnected from the net of inverter `clkinv_1_1` that it should drive, while DRC is clean. Unlike the DRC report above, this report names nets but gives no coordinates, so the Locator must

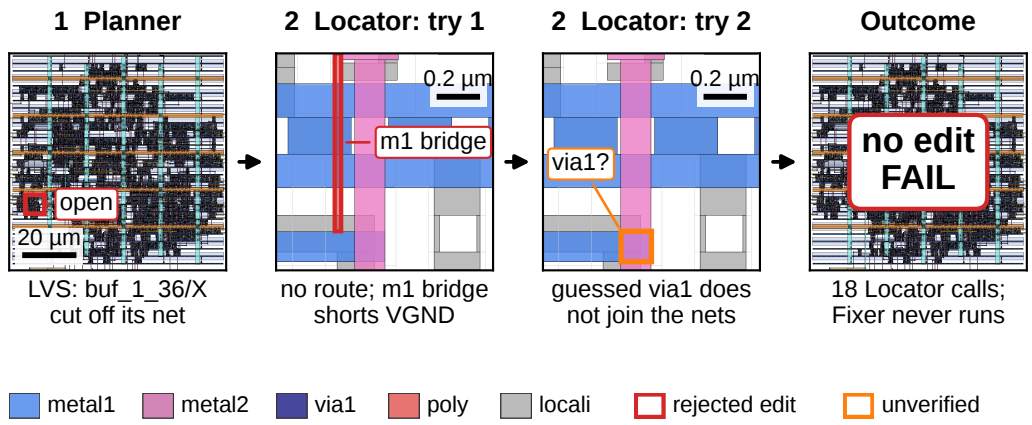
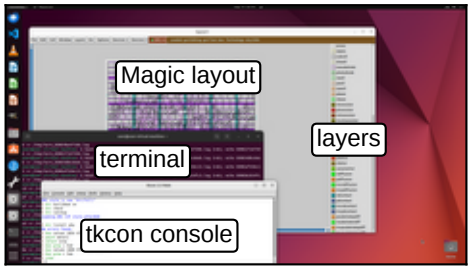


Figure 6: **Case study: an LVS open that is never localized** (task pv-0010, `gcd`, easy; gpt-5.6-sol backbone). (1) LVS reports the output of buffer `buf_1_36` disconnected from its net. (2) No route exists, and a metal1 bridge across the gap shorts VGND. (3) A hypothesized missing via1 does not join the two nets and is rejected. (4) The Locator exhausts its budget, so the layout is never edited.

find the break itself. Its routing tool finds no path between the two nets, and bridging the 0.82 μm metal1 gap between them would join them but also short the ground rail (VGND). The Locator then hypothesizes a missing via1 where a metal1 wire of the buffer's net crosses a metal2 wire it attributes to the inverter's net, but its own trials of that via disagree: one trial merges seven pins, most of them from other nets. The evidence check rejects both of its submissions because neither proposed bridge joins the two nets. After 18 model calls and 372K tokens the Locator runs out of steps; the Fixer and Reviewer never run, and the layout is returned unchanged. This is the dominant failure mode in Table 7: a fault that is never localized is never edited.

The same failure through the GUI. Figure 7 shows the same task run by Agent s3 in GUI mode, where the roles type their commands into a terminal and Magic's tkcon console on the desktop and read the console output from screenshots.

The agent's desktop



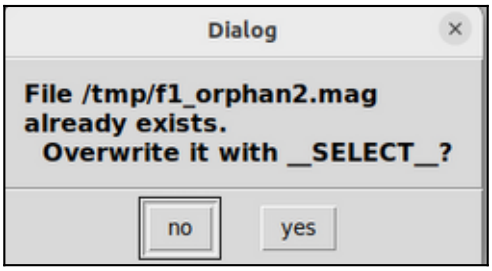
Magic, a terminal, and the tkcon console, driven by keyboard and mouse

1 Net lookup keeps failing

```
% eval box values [select bbox]
% select save /tmp/f1_orphan2
box values 763 4777 797 4811
% select net metal1
No paint of this type under the cursor/box
% select net metal2
No paint of this type under the cursor/box
```

select net at the pin label finds no paint, so the net dumps come out empty

2 A stray dialog stays open



a file-overwrite prompt opens and stays on screen until the run ends

3 Edit painted but never saved

```
% drc listall why
No errors found.
% box values 1854 4777 1900 4811
% paint metal1
% select area
% box grow c 5um
% box values 1800 4700 2000 4900
% box grow c 3um
% view
```

a metal1 bridge is painted by mistake and never written back to the GDS

Outcome: the GDS is never rewritten, so LVS still fails and the task is graded FAIL (layout unchanged) after 28.5 min, 174 GUI actions, 59 model calls, and 651K tokens.

Figure 7: **Case study: the same LVS open, operated through the desktop GUI** (task pv-0010, `gcd`, easy; Agent s3 in GUI mode, gpt-5.6-sol backbone). The agent works through a terminal and Magic's tkcon console on a 1920×1080 desktop. (1) `select net` at the pin label finds no paint, so the dumped net geometry is empty. (2) A file-overwrite dialog opens and stays on screen until the run ends. (3) A metal1 bridge is painted by mistake but never written back to the GDS, so the graded layout is unchanged.