

Invited: EDA-ENV: A Reinforcement Learning Environment for RTL-to-GDS Design with Independent Scoring

Amur Ghose
aghose@ucsd.edu
University of California San Diego
La Jolla, California, USA

Andrew B. Kahng
abk@ucsd.edu
University of California San Diego
La Jolla, California, USA

Yugesh Kothari
yugeshk@ergodex.ai
Ergodex AI, Inc.
Sunnyvale, California, USA

Umang Pandey
umang@ergodex.ai
Ergodex AI, Inc.
Sunnyvale, California, USA

Souktik Roy
souktik@ergodex.ai
Ergodex AI, Inc.
Sunnyvale, California, USA

Abstract

Language-model agents can implement and improve circuits using EDA tools. Evaluating their decisions requires a number of distinct RTL-to-GDS tasks. Agents can also modify the scripts that produce their reported metrics, so scoring must distinguish circuit improvements from changes to measurement. We present EDA-ENV, an end-to-end reinforcement learning environment with 103 tasks covering 12 categories of engineering work plus initial power, performance, and area (PPA) optimization problems across three process design kits (PDKs). Agents edit register-transfer-level (RTL) designs and implementation scripts under declared constraints and budgets. Independent scoring rebuilds submissions using protected libraries, evaluation constraints, and measurement code, permitting flow-wide edits while keeping scoring outside agent control. With the Kimi-k3 agent, an AES design on ASAP7 reduces effective clock period by 24.8%, with increases of 31.1% in estimated power and 1.6% in logical cell area; a power-grid repair reduces worst static voltage drop from 258 to 9.5 mV.

CCS Concepts

• **Computing methodologies** → **Artificial intelligence**; *Reinforcement learning*; • **Hardware** → **Physical design** (EDA).

Keywords

reinforcement learning, agentic EDA, RTL-to-GDS, physical design, terminal agents, benchmark, reproducibility

ACM Reference Format:

Amur Ghose, Andrew B. Kahng, Yugesh Kothari, Umang Pandey, and Souktik Roy. 2026. Invited: EDA-ENV: A Reinforcement Learning Environment for RTL-to-GDS Design with Independent Scoring. In *IEEE/ACM International Conference on Computer-Aided Design (ICCAD '26)*, November 08–12, 2026, San Jose, CA, USA. ACM, New York, NY, USA, 9 pages. <https://doi.org/10.1145/3831252.3848500>



This work is licensed under a Creative Commons Attribution 4.0 International License. ICCAD '26, San Jose, CA, USA
© 2026 Copyright held by the owner/author(s).
ACM ISBN 979-8-4007-2873-0/2026/11
<https://doi.org/10.1145/3831252.3848500>

1 Introduction

Implementing a design from register-transfer-level (RTL) source to a GDSII layout requires coordination of synthesis, floorplanning, placement, clock-tree synthesis, and routing. Decisions across these stages interact: a faster netlist can be harder to route, and a denser layout can worsen power delivery. Language-model agents perform RTL-to-GDS implementation work by reading tool reports, editing design files, and invoking EDA tools. Evaluating their decisions requires engineering tasks with physical outcomes, explicit constraints, and time and compute budgets.

Existing benchmarks evaluate physical-design skills and complete workflows [29], power, performance, and area (PPA) convergence and routing repair [30], or recovery across design stages [31]. Across these tasks, a successful edit must satisfy requirements beyond its immediate objective: e.g., grid repair must preserve timing and power constraints, and memory integration must preserve interfaces and read latency. A common environment is required to express these requirements across the full flow.

The freedom to edit implementation scripts creates a further evaluation problem. Rewards computed from agent-produced reports can increase without meeting the engineering objective, a form of reward hacking [48, 50]. For example, stale layouts or relaxed timing constraints can create apparent gains, or exposed reference solutions can bypass the task. Poorly defined metrics can also mislead a fixed scorer, e.g., counting filler cells in logical area rewards utilization changes without reducing logic.

The key insight is to separate the editable implementation project from the inputs and code that determine correctness and reward. An agent can then change the design and implementation flow while its result is measured against fixed engineering requirements.

We present EDA-ENV, an end-to-end RTL-to-GDS environment for agent-driven implementation, repair, and optimization using Yosys [1], OpenROAD [2], and OpenROAD-flow-scripts (ORFS) [3]. EDA-ENV supports complete flows from supplied RTL to final layouts, and tasks focused on individual flow stages. The scorer applies the agent's file changes to a fresh initial project, rebuilds the design, and checks it using inputs the agent cannot modify. We call this procedure *independent scoring* (Figure 1).

This paper makes three contributions:

- An end-to-end RTL-to-GDS environment with a common terminal interface for design and flow edits, and independent scoring to evaluate the resulting physical implementation.
- A collection of 103 tasks, with a construction and validation procedure covering 12 task families—categories of engineering

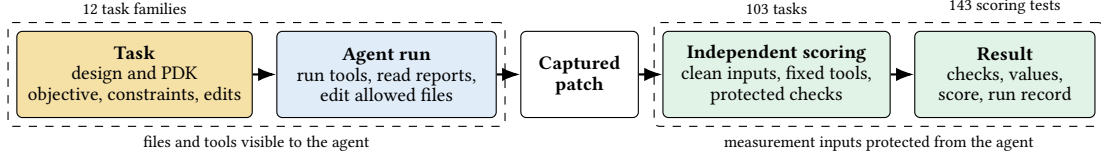


Figure 1: The EDA-ENV workflow. Tool reports guide agent edits, captured as a patch at submission. Independent scoring rebuilds the design and measures it against fixed task requirements.

work such as PDK transfer, power-grid repair, and memory integration—plus seven initial PPA optimization tasks.

- An evaluation of Kimi-k3 that measures task completion, physical improvements, and resource use, together with checks of the scorer’s measurement accuracy and repeatability.

Sections 3 and 4 describe the tasks and the independent scoring. Sections 5 and 6 validate the scoring procedure and evaluate the agent. See the released benchmark [4] for more details.

2 Related Work and Reference Data

Prior work develops optimization methods, agent interfaces, and reference implementations. In this section, we compare the stages and actions that these systems support and how they measure results, thus establishing requirements for evaluating agent decisions across the RTL-to-GDS flow.

2.1 Learning-based EDA and shared environments

Prior optimizers act on synthesis-pass sequences (DRILLS [6], ABC-RL [7], BoILS [8]), flow parameters (FIST [9]), or cell and macro placement (DREAMPlace [10], AutoDMP [11]). EDA-ENV additionally permits script, constraint, and module-connection edits for repairs that parameter tuning alone cannot express.

Reinforcement learning (RL) macro placers use graph observations [12], pixel representations and placement masks [13], or offline trajectory learning [14]. Intermediate placement metrics can misrank final PPA [15–17]. Gymnasium [18], CompilerGym [19], and ArchGym [20] separate algorithm interfaces from simulator or tool implementations. EDA-ENV applies this separation to terminal actions across the implementation flow.

2.2 Agents and benchmarks

EDA-ENV shares the command-line interface of Terminal-Bench [21] and the patch-based evaluation of SWE-bench [22] and HWE-Bench [23], with rewards tied to physical implementation. Previous works on tool interaction [24, 25], RTL generation [26, 27], and chip-design assistance [28] are concerned with how agents produce design changes; by contrast, EDA-ENV measures the physical outcomes of those changes.

Physical-design benchmarks already cover several forms of agent work. PDAgent-Bench [29] includes 353 conceptual and tool-based tasks plus a complete physical-design workflow. PostEDA-Bench [30] provides 145 tasks for PPA convergence

and design-rule-check (DRC) repair. CLOSER-Bench [31] studies budgeted recovery across RTL and physical-design stages. Our EDA-ENV covers transfer, block composition, power delivery, constraint repair, and scoped RTL changes, with design-specific requirements and protected measurement inputs.

Agentic physical-design systems [63] also differ in what the agent submits. ORFS-Agent [32] searches implementation parameters, and ChatEDA [61] and OpenROAD-Assistant [62] control the implementation flow through generated scripts. AuDoPEDA [33] tests OpenROAD source changes across designs. On the other hand, EDA-ENV evaluates task-specific design and flow patches. Table 1 contrasts this with routed PPA (ORFS-Agent), tool quality across designs (AuDoPEDA), and physical scores after task checks (EDA-ENV), i.e., pass rates are not directly comparable.

2.3 Design data and reference measurements

Many works have developed public collections of design data, e.g., the EPFL suite [34] supplies combinational circuits, and OpenABC-D [35] records sequences of synthesis transformations and their outcomes. CircuitNet [36] and EDALearn [37] provide physical design and cross-technology measurements. CircuitOps [38] provides graph interfaces; OpenROAD [2] and OpenLANE [39] provide implementation flows, with standardized flow metrics in METRICS2.1 [40]. HighTide [41] supplies 16 designs across three PDKs with incremental builds and support for porting, debugging, and memory generation.

The IEEE CEDA Design Automation Technical Committee (DATC) Robust Design Flow (RDF) supplies routed layouts for calibration. We use the RDF-2024 layouts and values from Section 3.2 and Table 3 of [42] to check timing units, clock-constraint interpretation, and repeatability. Rewards compare initial and submitted designs under identical evaluation conditions. RDF-2025 [43] and RDF-2026 [64] expand available calibrations and benchmarks.

2.4 Constraints and reward validity

Constrained Policy Optimization [46] and constrained Bayesian optimization [47] distinguish objective improvement from constraint satisfaction, as do our reward rules. Reward gaming [48] and reward-model overoptimization [49] concern mismatches between measured and intended objectives; reward tampering [50] changes the scoring rule or its inputs. We address misleading area

Table 1: Agent interfaces and evaluation scope, contrasting prior works with EDA-ENV. Different submitted changes and objectives prevent direct comparison of pass rates.

Work	Agent edits or output	Evaluation output	Implementation scope
SWE/HWE-Bench [22, 23]	Repository patch	Functional tests	Software or RTL repair
PDAgent-Bench [29]	Answers, scripts, design actions	Skill-specific checks and workflow results	Physical-design skills
PostEDA/CLOSER [30, 31]	Design and flow edits	PPA, DRC, or recovery	Post-flow and cross-stage closure
ORFS-Agent [32]	Flow parameters	Routed PPA	Implementation-recipe search
AuDoPEDA [33]	OpenROAD source patch	Quality of results across designs	Implementation-tool code
EDA-ENV	Problem-specific design and flow patch	Physical score after required checks	12 task families; Independent scoring

Table 2: The twelve task families contain 96 tasks. Seven initial AES, JPEG, and Ibex PPA tasks (F0) bring the total to 103. Included tasks have initial designs that fail the requirements and known solutions that pass, with both outcomes reproduced.

Family	Name	Tasks	Agent work	Required scoring checks
F1	Complete bring-up	10	Build a full recipe from supplied RTL	Function, fixed clock, utilization band, final route and DRC
F2	Trapped timing closure	10	Close a tighter clock under PPA constraints	WNS and area/power constraints relative to the initial design
F3	PDK transfer	6	Port the design to a destination PDK	Fixed RTL and clock, destination libraries, and final-layout checks
F4	Macro placement repair	9	Repair an intentionally damaged floorplan and route	Final layout database, timing, detailed-route DRC, and instance floor
F5	Macro integration	7	Replace behavioral storage with memory macros	Macro count, placement, signal/power connections, timing, DRC, testbench
F6	Power-grid repair	13	Replace a weak grid under PPA constraints	Declared supply net/voltage, static IR drop, timing, power, DRC
F7	Constraint repair	9	Repair clock, I/O, or clock-domain intent	Frozen evaluation SDC and final physical checks
F8	Flow repair	14	Diagnose a broken synthesis-to-route recipe	Complete final layout database, timing, DRC, and instance floor
F9	Block composition	2	Build and assemble reusable blocks	Required macro types, instance count, placement, and connectivity
F10	Constrained PPA	6	Minimize one PPA metric under constraints on the other two	Rebuilt initial-design values and declared PPA constraints
F11	Accelerator integration	3	Integrate an accelerator and its memory system	Structural instance check, macro inventory, instance floor, timing, DRC
F12	Logic depth and retiming	7	Change recipe, scoped RTL, or registers	Module equivalence or fixed-latency testbench, timing, DRC, instance floor

WNS: worst setup slack; DRC: design-rule checking; SDC: Synopsys Design Constraints; PDN: power-delivery network; IR drop: voltage loss in the power grid. An instance floor is a minimum permitted number of logical cell instances.

rewards by excluding physical-only cells, and tampering by protecting the scoring inputs and code. Section 5.1 examines these controls.

3 Task Design and Agent Interface

Each task combines an engineering problem, an editable project, and a scoring specification. We define these components before describing the task families, qualification process, and interface through which an agent works under a fixed resource budget.

3.1 Task, reference solution, and episode

A *task* specifies an engineering problem: an initial design project, an objective, engineering constraints, and the edits an agent may make. The unmodified project is the *starter*. A *reference patch* is a known solution used to validate the task and is withheld from the agent. A task is *qualified* when the starter fails its requirements and the reference patch passes them, with both outcomes reproduced by repeated independent scoring. An *episode* is one agent’s attempt to solve a task. It ends when the agent submits its changes or reaches its execution budget.

A versioned *task package* groups the instructions, initial project, edit policy, scoring specification, tests, and software-environment definition. The Docker image supplies the runtime tools and is one component of this package. Instruction *I* describes the objective and constraints; workspace *W* contains the starter; image *E* supplies the tools; *A* lists permitted edits; and *B* specifies the episode’s time and compute budget. The scorer *S* evaluates a submitted implementation, and the reward rule *R* maps the evaluation to a numerical reward. We represent the task by the tuple τ :

$$\tau = (I, W, E, A, B, S, R). \quad (1)$$

3.2 Task families and design context

An implementation recipe specifies the tool settings and scripts used at each stage. Table 2 groups tasks by the engineering work they require. The twelve generated task families contain 96 tasks; seven initial PPA optimization tasks (F0) bring the total to 103. A design can support several tasks with different faults or constraints. Each task family requires distinct edits and checks.

The PDK transfer family preserves RTL while adapting the implementation recipe to another technology. The flow uses ASAP7 [52], Nangate45 [66], and Sky130HD [67]. All three provide rules for extracting wire resistance and capacitance with OpenRCX in ORFS. PDK transfers supply the source recipe and working constraints, with both technologies’ physical and timing views in the installed ORFS tree.

Memory-macro integration matches a macro’s interface and cycle behavior to its physical and timing models, as supplied by compilers such as OpenRAM [58]. Block composition (F9) builds reusable hard blocks and assembles them in a top-level design. Power-grid repair supplies a weak power-delivery network (PDN) and a constraint on static voltage drop. Constraint repair supplies faulty clocks, I/O delays, or clock groups. Section 6.3 gives measured examples.

Table 3 lists the design–PDK pairs. The designs range from hundreds of placed instances to about one million in systems-on-chip (SoCs). Each instance is a placed copy of a standard cell or macro. BlackParrot [53] contributes nine Nangate45 tasks at 216k–226k instances, with 24 SRAMs and evaluation clock periods of 10, 8.5, or 7.65 ns. Eight Rocket–Gemmini [54] tasks use a 976k-instance SoC with an 8×8 array and ten SRAMs; they cover bring-up, PDN repair, flow repair, floorplanning, and accelerator integration. A ninth task replaces the array with a supplied 16×16 variant and its memory views, reaching 1.18 million instances. All nine use a 6.5 ns evaluation clock period: the preliminary periods of 5.778 ns (8×8) and 5.725 ns (16×16), each increased by 12%, round up to 6.5 ns.

Of the 116 original proposals, 14 did not qualify (Appendix A), leaving 102 tasks. Task 117 was added after the original roster, bringing the qualified set to 103, of which 53 are on ASAP7, 47 on Nangate45, and three on Sky130HD. Two additional Sky130HD-to-ASAP7 PDK transfer tasks did not qualify under the required KLayout DRC checks.

3.3 Task construction and qualification

The *task generator* creates the package from a design, PDK, task-family specification, and permitted edit list. It prepares the starter, instruction, scoring constraints, and tests, and packages the reference patch. The instruction is generated from the same scoring specification used by the evaluator. Package validation checks that the stated quantities, units, and constraints agree with that specification.

For qualification, scorer *S* independently scores the starter and the design obtained by applying the reference patch. The reference design must pass, and the starter must fail for the stated reason. A second run must reproduce both measurements exactly. Admission also checks ECP against the preliminary measurement, using starter timing or, when that comparison is unavailable, reference timing. Differences above 5% block admission. Checks that cannot be performed are reported as skipped.

As an example, task 081’s damaged grid has 14.7 mV worst static IR drop, and the reference repair reaches 11.3 mV. The task’s 13 mV constraint separates these measured cases.

Table 3: Designs, measured instances, and qualified tasks by family. Counts are at the top level, with each macro counted once. Transfers use the destination PDK. “Authored” identifies RTL circuits created by the authors; “assembled” identifies their connections of existing circuits.

Design	PDK	Instances	Tasks	Families
AES-128	ASAP7	10.6k–13.1k	18	F0 F1 F2 F3 F6 F7 F8 F9 F10
Multi-macro blocks mpb1–8 (authored, SRAMs)	ASAP7	1.8k–2.1k	12	F4 F6 F8
Ibex RV32	ASAP7	19.8k–48.9k	6	F0 F1 F3 F6 F7
JPEG encoder	ASAP7	53.9k–104k	4	F0 F2 F3 F7
16-bit multiplier (authored)	ASAP7	1.6k–1.7k	3	F1 F2 F12
Ethernet MAC	ASAP7	59.4k	2	F7
4× Ethernet MAC (assembled)	ASAP7	1.6k	2	F8 F9
TinyRISC-V	ASAP7	22.7k	1	F1
GCD	ASAP7	—	1	F8
S-box datapath (authored)	ASAP7	2.2k	1	F12
AES round variant (authored)	ASAP7	10.2k	1	F12
64-bit adder (authored)	ASAP7	1.8k	1	F12
4-tap FIR (authored)	ASAP7	1.6k	1	F12
BlackParrot tile (24 SRAMs)	Nangate45	216k–226k	9	F1 F2 F6 F7 F8
Rocket SoC + Gemini 8×8 (10 SRAMs)	Nangate45	976k	8	F1 F6 F8 F11
Rocket SoC + Gemini 16×16	Nangate45	1.18M	1	F11
JPEG encoder	Nangate45	82.8k	3	F0 F6
Systolic array (authored)	Nangate45	5.9k–8.8k	5	F1 F3 F10 F12
Ariane / CVA6 (133 SRAMs)	Nangate45	164k	4	F1 F4 F6
SweRV EH1	Nangate45	110k–159k	3	F2 F10
Ibex RV32	Nangate45	39.1k	1	F0
AES-128	Nangate45	13.7k–27.7k	2	F0 F3
Ethernet MAC	Nangate45	54.7k	2	F2
Packet store + SRAM (authored)	Nangate45	510	2	F5 F8
Key vault + SRAM, two variants (authored)	Nangate45	214–317	2	F5
I-cache + SRAMs, two variants (authored)	Nangate45	328–1.9k	2	F5
Scratchpad + SRAM, two variants (authored)	Nangate45	144–236	2	F5
SHA round (authored)	Nangate45	2.5k	1	F12
Elevator FSM	Sky130HD	165	1	F1
UART	Sky130HD	387	2	F1 F3
Total			103	

In EDA-ENV, agent images exclude reference answers, autotuner material, and numerical scoring files. Task constraints are validated using timing, area, power, or IR-drop measurements from the starter and reference implementations, as applicable. Public RTL may have appeared in a model’s training data; hence, evaluating generalization to unseen circuits requires private designs that are withheld from training.

3.4 Agent execution, workspace, and budgets

At episode start, the agent receives instruction I , the starter workspace, and the permitted edit list. It uses the installed Docker tools and observes files, reports, terminal output, exit codes, and remaining time. Macro tasks also supply interface, behavioral, physical, and timing models. Physical views use Library Exchange Format (LEF), and timing and power views use Liberty. The reference patch, evaluation constraints, fixed testbenches, and scorer are excluded from the image. The agent can read the task’s requirements but cannot edit their scoring implementation. Image-content checks detect accidental inclusion of protected files (Section 5).

The default edit policy accepts 237 named configuration variables and blocks 25 that can change protected inputs or bypass required flow stages. It also permits six path patterns along with selected stage scripts and hooks written in Tool Command Language (Tcl). Other edit policies add the working Synopsys Design Constraints (SDC) file, named RTL modules, or declared macro and wrapper files. The working SDC guides implementation, while the evaluation SDC fixes the measurement constraints. The scorer, standard-cell libraries, platform directory, build system, and scoring specification remain protected.

Instructions provide the flow command (orfs-agent-run), stages, and report paths. Agent actions range from brief inspections to hours-long flows, consuming unequal wall-clock time [51]. Layouts and reports reside outside the persistent project; patches capture source and recipe edits.

Image construction has a separate one-hour budget before agent execution. An episode has a wall-clock budget of four hours for 94 tasks and twelve hours for the nine Rocket–Gemmini tasks, with 8 CPUs, 16 GB of memory, and 30 GB of storage. A full Rocket–Gemmini flow takes seven to nine hours. Independent scoring has a separate two-hour budget. For Rocket–Gemmini, the scoring service allows fifteen hours, while the task bundles still declare a two-hour verifier timeout. Table 4 summarizes the task collection and experimental scope.

Table 4: Environment and experimental scope. Qualification and exclusion criteria are described in Section 3.3 and Appendix A.

Quantity	Value
Task families	12 task families; 116 original task proposals
Qualified tasks	116 original proposals – 14 exclusions + 1 later task (#117) = 103 qualified tasks on 3 PDKs
Kimi-k3 experiment	85 of 103 tasks evaluated; 72 pass every check
Agent budget	4 hours (94 tasks) or 12 hours (nine Rocket–Gemmini tasks); 8 CPUs; 16 GB of memory; 30 GB of storage
Scoring budget	Separate: 2 hours in task bundles, 15 hours in the Rocket–Gemmini scoring service
Scoring tests	143, all passing
Interaction data	7,132 steps and 7,556 tool calls in the selected runs
Released artifacts	Tasks, submissions, and evaluation tools (September 2026 checkpoint) [5]; benchmark website and task repository [4]

4 Independent Scoring and Reward

Independent scoring evaluates the implementation produced by the agent’s submitted patch, using protected evaluation inputs. It captures the edits, rebuilds the design, and checks task requirements through scorer S . Rule R converts the resulting checks and measurements into a reward.

4.1 Patch capture and design rebuild

The live agent runs as user agent with ownership of its workspace and home directory. At submission, root-owned capture code clears the result directory. It reads every allowed path from the workspace and records the difference from the starter as a patch.

A configuration parser checks variable names, value types, and argument syntax. It rejects file includes, make rules, line continuations, shell expansion, and protected assignments.

The scorer accepts up to 5 MB across 200 files. The 85 selected patches range from 10 to 319 lines (median 33), and touch one to four files (median one). None exceeds 12 KB. The scorer requires canonical Git paths whose parsed set matches the files changed by patch application. It rejects paths outside the task’s edit list, symbolic links, renames, and unsupported mode changes.

The scorer applies accepted edits to the starter, including assignment deletions, then builds from the declared RTL and stage scripts in an isolated directory. SHA-256 manifests check protected source files; synthesis-log checks compare source names, defines, top module, and parameters with the task declaration.

An unprivileged user runs the flow on eight CPUs without network access, using a container image fixed by its content hash. Each case has a separate directory, and reused containers clear prior work before starting. The live agent may access its model

API; the network restriction applies to the scoring flow. No generated layout, cached result, or measurement from the live agent is imported into scoring. The base image contains ORFS commit fb65d5b48 and OpenROAD 5c5380c49a, with fixed source patches. Starter, reference, and submission all use eight OpenROAD threads, matching the agent’s CPU allocation.

4.2 Timing, area, power, and routing

Effective clock period (ECP) is $ECP = T_{eval} - WNS$, where T_{eval} is the evaluation period and WNS is the worst setup slack. Slack is nonnegative when the critical path meets its requirement and negative otherwise. While OpenROAD reports this signed quantity as worst slack (WS), we use the more common WNS term in the above formula. The scorer uses the fixed evaluation SDC and extracted parasitics, even if the working flow uses another clock. If there are multiple clocks, the task must identify which clock is associated with the ECP. Reported clock periods use picoseconds for ASAP7 and nanoseconds for Nangate45 and Sky130HD; each table labels its units.

The measurement program opens the final OpenDB (ODB) layout and loads scorer-owned Liberty views and the evaluation SDC. ASAP7 uses fast-fast (FF) Liberty models at 0.77 V and 0°C for the RVT, LVT, SLVT, and SRAM cell families. Nangate45 uses typical models at 1.10 V and 25°C; Sky130HD uses TT models at 1.80 V and 25°C. Macro tasks declare additional libraries. Parasitics come from the platform’s RC-extraction rules, which all three PDKs provide. Clocks are propagated for setup and hold analyses. The task’s scoring specification sets pass/fail requirements for these measurements.

Logical cell area is the sum of the areas of the placed cell instances after excluding physical-only cells (filler and tap cells, including tap cells used as endcaps). It is separate from die area and utilization, and it includes macro area when macros are present. Logical instance count uses the same exclusions and counts each macro once. The scorer also measures timing and power on the final layout, including when area is the optimization objective. OpenSTA [55] reports total power using default vectorless activity assumptions rather than a simulated workload trace.

The current scorer attempts a fresh OpenROAD check_drc on the final ODB and compares it with the detailed-route violation metric. It uses the larger count when both are available. If the fresh check is unavailable, it records that limitation and uses the stage metric. Absence of both measurements causes rejection. Variables that skip detailed routing or metric reporting are blocked.

Workspace Liberty files are hashed before and after flow execution. A change invalidates the run. This detects modification during the flow, but does not authenticate an arbitrary replacement library.

Static IR drop analysis runs for tasks that name a supply net and a voltage-drop constraint. A pass needs the measured worst static IR drop below that threshold together with the task’s timing and power conditions. This analysis is at present a coarse check: OpenROAD’s static solver runs on the extracted grid with vectorless power, and the power grids of the open platforms are not calibrated to power-integrity targets.

4.3 Functional and structural checks

Required evidence depends on the permitted edit and has associated limitations. For example, fixed-RTL tasks protect source text with hashes. Tasks that permit scoped combinational edits use Epy [56], mapping inferred memories to logic and requiring every Boolean satisfiability (SAT) equivalence partition to pass against a

fixed reference. For the S-box task, Epy checks equivalence to the fixed 256-entry function. AES-round and SHA-round tasks restrict changes to mixcolumns.v and shacomp.v, with fixed wrappers and interfaces. Multiplier and adder tasks permit recipe changes while retaining fixed RTL.

Retiming moves registers across combinational logic while preserving specified behavior across cycles [57]. The four-tap finite impulse response (FIR) filter has coefficients 13, 7, 5, and 3 and a four-cycle input-to-output latency. A protected Icarus Verilog testbench checks the arithmetic outputs and latency on more than 2,000 samples after reset. Register movement is permitted if these checks pass. Thus, the testbench provides evidence for the task’s specified behavior, but is not an exhaustive sequential-equivalence proof. All logic-depth tasks also require nonnegative evaluation-clock slack, zero routing violations, and at least 30% of the starter’s logical instance count. For example, the S-box starter has 320.3 ps ECP against a 312 ps evaluation period. A passing recipe or RTL change must improve timing while also preserving function.

Memory integration requires agreement between behavior and physical connections. A fixed testbench checks width, depth, masks, ports, and cycle behavior. ODB queries check required macro identities and counts, legal placement, connected signal terminals, and supply terminals on power/ground nets. Accelerator integration adds a structural check of the accelerator instance and each port’s connection to its reference net, while the RTL and the tile logic outside the instance remain fixed. This check does not establish correctness of the accelerator’s architectural protocol.

Constraint repair uses fixed evaluation clocks, I/O budgets, and declared clock-domain relationships. This prevents a submission from passing solely by relaxing its working SDC. These checks establish implementation timing under the task’s specification. However, they do not establish whether a new clock-domain exception follows from synchronizers or interface behavior. The current tasks therefore use authored constraint faults.

Macro-placement tasks do not measure halo, channel width, or pin access separately. Power-grid tasks require a completed static analysis satisfying the voltage-drop constraint, alongside timing and power checks. However, they do not count straps and vias or prove I/O-pad coverage. Instance-count constraints detect gross logic removal but do not replace final-netlist equivalence.

4.4 Objectives, constraints, and reward

Tasks use one of two reward rules. A *completion task* asks the agent to satisfy all required conditions. An *optimization task* also names a physical metric to minimize, such as effective clock period, logical area, or total power. For example, a timing objective may permit area and power increases of at most 3% and 5%. The scorer then sets those constraints to 1.03 times the rebuilt starter’s area and 1.05 times its power. For such relative constraints, the scorer rebuilds the starter and submitted design under the same settings in separate directories. If the required starter build fails, the comparison is invalid and does not count as an agent failure.

Let g_j denote required condition j , with indicator $1[g_j]$ equal to one when it holds and zero otherwise. Let G indicate that all conditions pass. For measured objective $m > 0$ and unit-conversion factor s , the optimization reward r is

$$G = \prod_j 1[g_j], \quad r = \begin{cases} s/m, & G = 1, \\ 0, & G = 0. \end{cases} \quad (2)$$

Table 5: Evaluation failure modes and corresponding controls. The last column lists the required regression checks.

Property	Failure mode	Control	Required regression check
Measurement validity	Missing threshold-voltage views produce unbound cells and nonphysical slack Hierarchical builds omit macro timing models Physical-only cells make area depend on utilization Missing routing metrics appear to report zero violations	Load all allowed Liberty views; reject nonphysical timing Generate, ship, and bind <code>build_macros</code> outputs Sum logical instance areas, excluding filler and tap cells Require a final routing metric and block route-skipping variables	Missing-cell and extreme-WNS inputs rejected Block-Liberty generation and binding Area exclusions and instance-floor enforcement Absent metrics rejected; skip variables blocked
Submission integrity	Candidate changes remain outside the captured workspace Allowed files are omitted from the copy list Configuration syntax differs between the agent and scorer Rebuilding restores deleted assignments Patch headers conceal paths from the whitelist parser	Define persistent paths and copy rules Derive capture paths from the allowed edit list Parse typed assignments using the declared grammar Record deleted assignments explicitly and preserve their deletion Require canonical headers and agreement between parsed and changed paths	Empty-patch recording and capture of allowed files Every allowed path copied Shell, make, continuation, and accepted-argument cases Deleted assignments remain absent Noncanonical headers rejected; permitted new-directory patches accepted
Execution isolation	Runtime images expose reference solutions Reused containers retain stale build products Concurrent runs share output directories Concurrent scorings retrieve another task's report Privileged agent execution exposes protected capture paths	Remove reference files and inspect image contents Use a clean directory for each case Allocate a unique directory per run Retain reports by problem name Use an unprivileged agent, root-only copying, and a cleared result directory	No reference solution or clean configuration in the agent image Rebuild after container reuse Concurrent-run output isolation Concurrent-run report isolation Trial episode under the unprivileged agent user

The factor s converts inverse period to GHz, inverse cell area to mm^{-2} , or inverse power to W^{-1} . For completion tasks, we set the reward to the indicator G . Missing or nonphysical measurements fail the corresponding check.

The collection contains 30 optimization tasks and 73 completion tasks. Optimization covers the initial PPA tasks, timing closure, constrained PPA, and logic depth; the other task families use the completion reward. The objective and all constraints are declared before the episode starts.

4.5 Result record and error handling

Each artifact result links its score to measured and starter values, failed conditions, package checksum, patch metadata, evaluation log, agent identity, resource totals, and exceptions. Rejections identify the offending file, value, or physical condition. Missing measurements fail, while a failed required starter or reference build invalidates the scoring call.

5 Validation of Independent Scoring

Scorer validation precedes agent evaluation. We examine failures in measurement, patch capture, and execution isolation, together with the regression checks developed from them. Published layouts, along with repeated measurements and rebuilds, provide checks on measurement accuracy and repeatability.

5.1 Evaluation failure modes and regression checks

Independent scoring requires valid measurement inputs, faithful patch reconstruction, and isolated executions. Missing timing views or routing metrics invalidate scores; incomplete capture or inconsistent parsing can alter the evaluated design; shared state or privileged access compromises isolation. Table 5 pairs these failures with controls and regression tests. All 143 regression tests pass in the September 2026 version.

Excluding physical-only cells changes whether an area reduction meets the task's requirement. For the AES/ASAP7 development example in Table 6, including physical-only cells gives an area change from 2,530 to 1,360 μm^2 . Logical area instead falls from 1,465 to 944 μm^2 , a 35.6% reduction. For an excluded Ibex/Nangate45 candidate, the area reduction falls from 1.22% to 0.08%, below the task's required improvement. Thus, the AES reduction meets the logical-area requirement, while the Ibex reduction does not.

Ten additional validation episodes passed under the unprivileged agent identity and patch checks. An audit of patches and tool calls from the 85 selected episodes plus these ten runs found no patch-header bypass or writes to protected capture or measurement paths, which are the two exploit classes examined.

5.2 Measurement checks and repeatability

We test ECP measurements on all 18 routed RDF-2024 reference layouts [42, Sec. 3.2, Table 3]. Sixteen values agree with the published values within 0.9%.¹

The JPEG/Nangate45 starter recipe for task 002 shifts ECP by 2.7% between eight and sixteen threads, which motivates fixing the thread count across comparisons. Nine RDF layouts reproduced their ECP on a second measurement. Six AES starter rebuilds across three days and two images returned 387.5975 ps at eight threads.

6 Experimental Results

Our main experiment evaluates Kimi-k3 through task completion, resource use, and physical changes to the submitted designs. After defining the run set, we examine success across task families, representative repairs, and PPA trade-offs. Earlier development runs are reported separately. Task descriptions and reproduction instructions accompany the benchmark [4].

6.1 Run selection and resource use

The experiment uses `kimi-cli` with `moonshot/kimi-k3` on 85 of the 103 qualified tasks, with one selected episode per task. The remaining 18 tasks are later-qualified BlackParrot and Rocket-Gemmini tasks that expanded the collection beyond this experiment's task set; they are not failed qualification candidates. These include all three F11 accelerator-integration tasks, so F11 has no evaluated runs in Figure 2. For tasks affected by instruction, capture, or infrastructure defects, we use corrected reruns or rescored intact submissions. The ten additional validation episodes in Section 5.1 are excluded from these totals.

The selected episodes contain 7,132 interaction steps, 7,556 tool calls, and 4,594 submitted patch lines. All patches are nonempty. Elapsed time totals 114.2 hours, with a 31.9-minute median and a 275.4-minute maximum, excluding later rescoring. Eleven agent-execution intervals exceed the nominal 240-minute limit. We retain those records in the descriptive results, but cannot use these 85

¹The two commercial-flow JPEG layouts agree after applying RDF's clock-bound, zero-I/O-delay constraints.

Table 6: Representative implementation measurements and starter-relative trade-offs for selected task IDs. Area is logical cell area; power is vectorless total power. “Development” rows are outside the Kimi-k3 experiment. A dash denotes an unavailable starter-relative measurement. Closure labels identify task specifications rather than measured gains. VT denotes threshold voltage. A VT swap selects a different cell family; LVT identifies the low-VT family selected in task 013. See FrontierEDA [4] for broader benchmarking across models.

Run set	Design / PDK	Objective	Starter → submission	Objective change	Area	Power	Submitted changes / check
Kimi 001	AES / ASAP7	ECP, no caps	387.6→291.5 ps	−24.8%	+1.6%	+31.1%	VT swap to faster cells; DRC 0
Kimi 037	S-box pipe / ASAP7	ECP, no caps	320.3→176.5 ps	−44.9%	+20.2%	+39.7%	VT swap, timing-driven placement; Eeq equivalent
Kimi 013	AES close-5 / ASAP7	ECP, area +3%, power +5%	387.6→354.2 ps	−8.6%	−11.2%	+0.2%	LVT cells, synthesis clock, lower utilization; caps met
Kimi 097	AES 3-objective / ASAP7	ECP under area and power caps	387.6→327.7 ps	−15.5%	−6.3%	+2.0%	VT swap, floorplan; caps met
Kimi 092	Ethernet MAC close −5% / Nangate45	ECP under caps	2.907→2.779 ns	−4.4%	+0.1%	−1.1%	synthesis clock target; repair; DRC 0
Kimi 093	Ethernet MAC close −15% / Nangate45	ECP under caps	2.897→2.811 ns	−2.9%	+0.2%	0.0%	synthesis clock target; DRC 0
Kimi 075	Adder64 depth / ASAP7	ECP, no caps	369.2→222.7 ps	−39.7%	+4.5%	+21.4%	adder architecture map, VT swap; DRC 0
Kimi 032	Ariane bring-up / Nangate45	Completion	stub→clean layout at 3.6 ns	—	—	—	floorplan and macro placement; pass
Kimi 009	AES / Nangate45	Logical area	19,245→16,378 μm^2	−14.9%	—	−28.2%	area-mode synthesis, higher utilization; ECP +2.6%, DRC 0
Development	AES / ASAP7	ECP, no caps	387.6→288.3 ps	−25.6%	—	—	pass; 3.469 GHz
Development	JPEG / Nangate45	ECP, no caps	1.138→0.910 ns	−20.0%	+71.9%	+69.6%	pass
Development	AES / ASAP7	Logical area	1,465→944 μm^2	−35.6%	—	—	pass; DRC 0

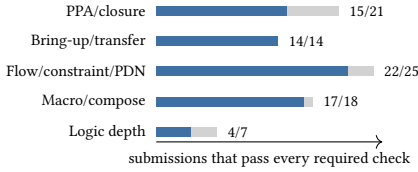


Figure 2: Submissions in the 85-run kimi-k3 experiment that pass every required check, by task-family group. Blue shows passes and gray shows failures. Groups: PPA/closure = F0, F2, F10; bring-up/transfer = F1, F3; flow/constraint/PDN = F6, F7, F8; macro/compose = F4, F5, F9; logic depth = F12.

selected episodes for a strict equal-budget comparison. Token usage totals 464.47 million input tokens, including 456.36 million cached tokens, and 2.06 million output tokens.

Development runs are eleven earlier episodes collected while the prompts and scorer were changing. Their PPA examples illustrate the consequences of different task constraints (Section 6.4); they are excluded from Kimi completion counts and resource totals.

Table 7: Episode time by task-family group, excluding later rescoring. The interval spans the 25th to 75th percentiles.

Task-family group	Runs	Episode time		
		Median (min)	25th–75th (min)	Total (h)
PPA / closure	21	195.9	150.2–242.5	69.0
Bring-up / transfer	14	12.0	7.3–22.7	5.6
Flow / constraint / PDN	25	26.4	13.2–40.0	19.0
Macro / compose	18	18.4	10.7–25.4	10.1
Logic depth	7	88.6	84.2–96.5	10.5
All selected episodes	85	31.9	—	114.2

PPA and closure account for 21 of 85 episodes but 60.4% of elapsed time (Table 7). The thirteen unsuccessful episodes consume 26.2 hours, or 22.9% of the total. Optimization thus dominates elapsed time, and unsuccessful attempts consume nearly a quarter of it. These costs include agent deliberation and tool execution.

6.2 Success across task families

Kimi passes 72 of 85 tasks, or 84.7%. Figure 2 groups the task families into five categories. Kimi passes all 14 bring-up and transfer tasks, while pass rates are lower in the PPA/closure and logic-depth groups. A logic-depth pass can result from recipe changes without an RTL edit.

Table 8 distinguishes physical failures from interface errors, with associated task IDs. Task 090 timed out and received zero reward. The retained record does not give a detailed cause.

Table 8: Failed episodes by cause.

Cause	Count	Task IDs
Setup timing miss (10–131 ps)	5	059, 073, 080, 082, 098
Unsupported configuration variable	5	004, 014, 068, 069, 084
Power constraint violated	1	031
Wrapper fails to elaborate	1	088
Timeout (detailed cause unavailable)	1	090

6.3 Repairing designs under fixed constraints

Repair tasks require correcting faults while satisfying the remaining requirements, including timing under fixed evaluation constraints.

In task 034 (AES/ASAP7), the starter increases the stock grid’s strap spacing on M5 and M6 by a factor of 4, raising worst static IR drop from 19.3 to 258 mV. The reference restores the stock pitch and its 19.3 mV drop. Kimi’s denser-grid submission reaches 9.5 mV, below the 25 mV limit. The sparse-grid starter has 390.3 ps ECP and 80.7 mW total power. Kimi’s submission also selects LVT cells and enables placement timing repair, reaching 373.7 ps ECP and 78.8 mW total power. Both meet the 400 ps timing limit and the power cap of 1.05 times starter power.

In task 081 (mpb1/ASAP7), Kimi reduces worst static IR drop from 14.7 to 5.4 mV, below the 13 mV limit. Area remains at 5,849 μm^2 , and power changes from 57.3 to 57.4 mW. Task 083 provides a larger Nangate45 example: Ariane’s worst static IR drop falls from 8.78 to 1.8 mV, satisfying its 6.6 mV constraint, with reported power unchanged at 217 mW.

Kimi’s repair of the Ethernet implementation constraints in task 079 meets the fixed evaluation timing requirement. The starter removes clock groups and swaps the Wishbone and MII clock periods, but its qualification run has −60.9 ps worst setup slack under the fixed evaluation SDC. Kimi’s submitted repair reaches +14.5 ps worst setup slack under the same constraints. On the other hand, for the similar task 073, Kimi’s submission still fails with −42.8 ps worst setup slack: the fixed evaluation SDC distinguishes a successful repair from a submission that still misses the timing requirement.

For task 084 (four-tap FIR retiming), the reference reaches 365.1 ps ECP from 528.9 ps while passing the four-cycle functional test. The reference patch passes all required checks, but Kimi’s submission is rejected by the configuration parser.

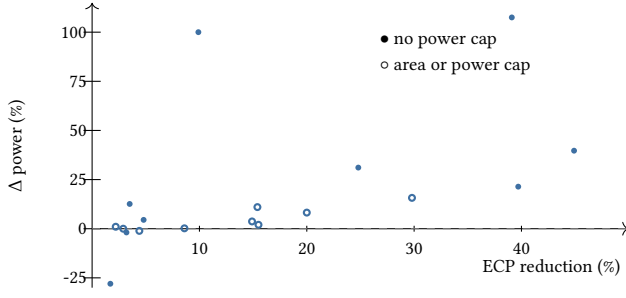


Figure 3: ECP reduction and vectorless power change for all 18 successful period-optimization tasks, relative to their starters. Filled points have no power cap; open points carry an area or power constraint.

Table 9: Excluded task proposals and measured causes. Proposal labels follow the family ledger. NG45: Nangate45.

Proposal; design / PDK	Qualification outcome
F2.2–2.3; Ibox / ASAP7 F2.7; JPEG / ASAP7	Reference sweep fails to meet clock periods tightened by 5% and 15%. Closing needs +9.4–10% power against a +5% cap; the tested configuration near the power cap still misses timing by 29 ps.
F7.4, F7.8; JPEG / ASAP7 F9.3–9.4; Ariane / NG45	Fault shifts ECP by 6 ps; starter and reference both pass. Cache block fails placement at 45% utilization and global routing at 30%.
F10.4; systolic / NG45 F10.5; JPEG / NG45	Tested timing repair needs +9.8% area, exceeding the area cap. Fastest tested recipe adds 48% area; tested synthesis scripts targeting lower delay worsen ECP.
F10.8; SweRV / NG45 F10.9–10.11; Ethmac / NG45	Tested timing repair adds 4.7% area against a 2% cap. Tested repairs add 1.9–2.1% area and 37% power to reduce ECP by 6–13 ps; looser variants duplicate qualified tasks.
Area; Ibox / NG45	Logical-area gain is 0.08%, below the required improvement; including physical-only cells gives 1.22%.

6.4 PPA improvements and trade-offs

Table 6 reports example PPA improvements and their costs under the respective tasks’ engineering constraints. Figure 3 covers all 18 successful period-optimization tasks; the other 54 passes comprise one area task and 53 completion tasks.

AES 001 and S-box 037 select faster threshold-voltage cells and change the implementation recipe. The S-box RTL is unchanged and passes Eeq checks. Under closure constraints, AES 013 reduces area by 11.2% with a 0.2% power increase. However, AES tasks 001 and 013 also have different timing requirements, so this comparison does not isolate the effect of the area and power constraints.

The development examples show why an improvement must be read against its task constraints. The best AES development run reaches 288.3 ps, or 3.469 GHz. A JPEG speed-only run reaches 0.910 ns with power increasing from 0.585 to 0.992 W and area from 107.8k to 185.4k μm^2 , with both increases permitted by the task. Kimi’s power-constrained AES submission in task 031 reaches 337.4 ps but receives zero reward because 0.112 W exceeds its 0.105 W power constraint.

For comparison, RDF-2025 [43] reports 0.927 ns for the best of 1,000 ORFS tuning runs and 0.632 ns for Aprisa on JPEG/Nangate45, versus 0.9096 ns in our development run. However, this is not an apples-to-apples comparison, given the differences in tools, extraction, and libraries. Rewards compare each starter and submission under common scoring conditions.

7 Scope and Limitations

Three limits affect interpretation of what we report: experimental coverage, verification assumptions, and agent observations.

The experiment evaluates one agent configuration on 85 tasks, not RL training effectiveness or model ranking. Episodes span package revisions and lack repeated-run uncertainty estimates; 18 later-qualified SoC tasks remain unevaluated. Controlled comparisons

require fixed task versions, enforced budgets, tests on designs excluded from training, and repeated runs [59]. Related RTL variants must remain in the same data split, so that training and test sets do not contain nearly identical circuits.

The physical checks use open timing and extraction tools, vectorless power, OpenROAD routing checks, and static IR analysis. They only support comparison under the declared setup, and do not rise to any level of “sign-off”. Functional coverage differs across source hashes, scoped equivalence, and testbenches; this enables the passing submissions 009 and 099 to change synthesis or flow code without a final-netlist equivalence check. The host, container runtime, image registry, and scoring service remain trusted components.

The interface lacks standard spatial observations of macro adjacency, pin access, congestion, and grid coverage. MaskPlace’s rasters [13], CircuitOps’ graphs [38], and PowerNet’s dynamic IR-drop features [60] are possible representations for these observations. However, we have not evaluated these representations in EDA-ENV.

8 Conclusion

EDA-ENV provides an end-to-end RTL-to-GDS environment with 103 tasks across three PDKs, a common terminal interface, and independent scoring of submitted edits. The tasks span PPA optimization, implementation repairs, memory integration, block composition, and RTL restructuring. Kimi-k3 passes 72 of 85 selected tasks, demonstrating both physical improvements and repairs under the declared engineering constraints.

We highlight three lessons. First, timing gains must be assessed with their area and power costs. Second, interface errors merit separate diagnosis: five of thirteen failed episodes propose an unsupported configuration variable. Third, full-flow runtime limits the number of attempts on large designs, making early diagnosis and implementation choices consequential.

Ongoing work includes evaluation of the 18 later-qualified SoC tasks, model comparisons using private held-out instances and repeated runs, and spatial observations for macro and grid tasks. We also note that recorded patches, trajectories, and measurements support offline learning [44] and reward-conditioned action prediction [45]. See the benchmark [4] and archived artifact [5] for supporting materials.

Acknowledgments

AI usage statement. The authors used OpenAI Codex for language editing, repository inspection, arithmetic checks, and $\text{\LaTeX}$ preparation.

A Qualification and Artifact Notes

Table 9 lists 14 excluded task proposals. Failures apply to the tested edits and reference construction, not to every possible solution. The reference layouts for the two additional Sky130HD-to-ASAP7 transfer proposals also failed KLayout checks [65]. No qualified task therefore requires KLayout checking; evaluation uses the OpenROAD routing checks in Section 4.

B Artifact Release

The archived artifact [5] contains 85 task packages, final submissions, and evaluation tools. Reproduction instructions accompany the benchmark [4].

References

- [1] C. Wolf *et al.*, “Yosys—A free Verilog synthesis suite,” in *Proc. Austrochip*, 2013.
- [2] T. Ajayi *et al.*, “Toward an open-source digital flow: First learnings from the OpenROAD project,” in *Proc. DAC*, 2019, Art. no. 76.
- [3] The OpenROAD Project, “OpenROAD-flow-scripts: An autonomous RTL-to-GDSII flow,” 2026. [Online]. Available: <https://github.com/The-OpenROAD-Project/OpenROAD-flow-scripts/tree/fb65d5b48>
- [4] Ergodex AI, Inc., “FrontierEDA,” 2026. [Online]. Website: <https://frontiereda.com>; accompanying repository: <https://github.com/Ergodex-Labs/eda-env-iccad-26>
- [5] A. Ghose *et al.*, “Invited: EDA-env: A reinforcement learning environment for RTL-to-GDS design with independent scoring,” Zenodo, 2026, doi:10.5281/zenodo.22990248.
- [6] A. Hosny *et al.*, “DRiLLS: Deep reinforcement learning for logic synthesis,” in *Proc. ASP-DAC*, 2020.
- [7] A. B. Chowdhury *et al.*, “Retrieval-guided reinforcement learning for Boolean circuit minimization,” in *Proc. ICLR*, 2024.
- [8] A. Grosnit *et al.*, “BOiLS: Bayesian optimisation for logic synthesis,” in *Proc. DATE*, 2022, pp. 1193–1196, doi:10.23919/DATES4114.2022.9774632.
- [9] Z. Xie *et al.*, “FIST: A feature-importance sampling and tree-based method for automatic design flow parameter tuning,” in *Proc. ASP-DAC*, 2020, pp. 19–25.
- [10] Y. Lin *et al.*, “DREAMPlace: Deep learning toolkit-enabled GPU acceleration for modern VLSI placement,” in *Proc. DAC*, 2019, Art. no. 117, doi:10.1145/3316781.3317803.
- [11] A. Agnesina *et al.*, “AutoDMP: Automated DREAMPlace-based macro placement,” in *Proc. ISPD*, 2023, pp. 149–157, doi:10.1145/3569052.3578923.
- [12] A. Mirhoseini *et al.*, “A graph placement methodology for fast chip design,” *Nature*, vol. 594, pp. 207–212, 2021, doi:10.1038/s41586-021-03544-w.
- [13] Y. Lai *et al.*, “MaskPlace: Fast chip placement via reinforced visual representation learning,” in *Advances in Neural Information Processing Systems*, vol. 35, 2022, pp. 24019–24030.
- [14] Y. Lai *et al.*, “ChiPFormer: Transferable chip placement via offline decision transformer,” in *Proc. ICML*, PMLR vol. 202, 2023, pp. 18346–18364.
- [15] C.-K. Cheng *et al.*, “Assessment of reinforcement learning for macro placement,” in *Proc. ISPD*, 2023, pp. 158–166, doi:10.1145/3569052.3578926.
- [16] C.-K. Cheng *et al.*, “An updated assessment of reinforcement learning for macro placement,” *IEEE Trans. Comput.-Aided Des. Integr. Circuits Syst.*, early access, 2025, doi:10.1109/TCAD.2025.3644293.
- [17] Z. Wang *et al.*, “Benchmarking end-to-end performance of AI-based chip placement algorithms,” in *Advances in Neural Information Processing Systems*, vol. 38, 2025.
- [18] M. Towers *et al.*, “Gymnasium: A standard interface for reinforcement learning environments,” arXiv:2407.17032, 2024.
- [19] C. Cummins *et al.*, “CompilerGym: Robust, performant compiler optimization environments for AI research,” in *Proc. CGO*, 2022, pp. 92–105, doi:10.1109/CGO53902.2022.9741258.
- [20] S. Krishnan *et al.*, “ArchGym: An open-source gymnasium for machine learning assisted architecture design,” in *Proc. ISCA*, 2023, Art. no. 14, doi:10.1145/3579371.3589049.
- [21] M. A. Merrill *et al.*, “Terminal-Bench: Benchmarking agents on hard, realistic tasks in command line interfaces,” arXiv:2601.11868, 2026.
- [22] C. E. Jimenez *et al.*, “SWE-bench: Can language models resolve real-world GitHub issues?” in *Proc. ICLR*, 2024.
- [23] F. Cui *et al.*, “HWE-Bench: Benchmarking LLM agents on real-world hardware bug repair tasks,” arXiv:2604.14709, 2026.
- [24] S. Yao *et al.*, “ReAct: Synergizing reasoning and acting in language models,” in *Proc. ICLR*, 2023.
- [25] J. Yang *et al.*, “SWE-agent: Agent-computer interfaces enable automated software engineering,” in *Advances in Neural Information Processing Systems*, vol. 37, pp. 50528–50652, 2024, doi:10.52202/079017-1601.
- [26] M. Liu *et al.*, “VerilogEval: Evaluating large language models for Verilog code generation,” in *Proc. ICCAD*, 2023, doi:10.1109/ICCAD57390.2023.10323812.
- [27] Y. Lu *et al.*, “RTLML: An open-source benchmark for design RTL generation with large language model,” in *Proc. ASP-DAC*, 2024, pp. 722–727.
- [28] M. Liu *et al.*, “ChipNeMo: Domain-adapted LLMs for chip design,” arXiv:2311.00176, 2023.
- [29] Q. Li *et al.*, “PDAGent-Bench: Characterizing, grounding, and architecting LLM/VLM agents for VLSI physical design,” arXiv:2606.17253, 2026.
- [30] P. Liu *et al.*, “Bridging the last mile of circuit design: PostEDA-Bench, a hierarchical benchmark for PPA convergence and DRC fixing,” arXiv:2605.06936, 2026.
- [31] P. Zhou *et al.*, “CLOSER-Bench: Evaluating budgeted cross-stage design closure for hardware agents,” arXiv:2607.16632, 2026.
- [32] A. Ghose *et al.*, “ORFS-Agent: Tool-using agents for chip design optimization,” in *Proc. MLCAD*, 2025, pp. 1–13, doi:10.1109/MLCAD65511.2025.11189204.
- [33] A. Ghose *et al.*, “Automated QoR improvement in OpenROAD with coding agents,” in *Proc. ICCAD*, 2026, doi:10.1145/3831252.3834240.
- [34] L. Amarú *et al.*, “The EPFL combinational benchmark suite,” in *Proc. IWLS*, 2015.
- [35] A. B. Chowdhury *et al.*, “OpenABC-D: A large-scale dataset for machine learning guided integrated circuit synthesis,” arXiv:2110.11292, 2021.
- [36] Z. Chai *et al.*, “CircuitNet: An open-source dataset for machine learning in VLSI CAD applications with improved domain-specific evaluation metric and learning strategies,” *IEEE Trans. Computer-Aided Design of Integrated Circuits and Systems*, vol. 42, no. 12, pp. 5034–5047, 2023.
- [37] J. Pan *et al.*, “EDALearn: A comprehensive RTL-to-signoff EDA benchmark for democratized and reproducible ML for EDA research,” in *Proc. ICCAD*, 2024, doi:10.1145/3676536.3697116.
- [38] R. Liang *et al.*, “CircuitOps: An ML infrastructure enabling generative AI for VLSI circuit optimization,” in *Proc. ICCAD*, 2023.
- [39] M. Shalan *et al.*, “Building OpenLANE: A 130nm OpenROAD-based tapeout-proven flow,” in *Proc. ICCAD*, 2020, pp. 1–6, doi:10.1145/3400302.3415735.
- [40] J. Jung *et al.*, “METRICS2.1 and flow tuning in the IEEE CEDA Robust Design Flow and OpenROAD,” in *Proc. ICCAD*, 2021, pp. 1–9.
- [41] B. Goldblatt *et al.*, “HighTide: An agent-curated open-source VLSI benchmark suite,” arXiv:2606.04126, 2026.
- [42] V. A. Chhabria *et al.*, “Strengthening the foundations for IC physical design and ML EDA research,” in *Proc. ICCAD*, 2024, Art. no. 199, doi:10.1145/3676536.3697136.
- [43] V. A. Chhabria *et al.*, “Invited: IEEE DATC RDF-2025: Enabling an EDA research ecosystem,” in *Proc. ICCAD*, 2025, pp. 1–9, doi:10.1109/ICCAD66269.2025.11240743.
- [44] J. Fu *et al.*, “D4RL: Datasets for deep data-driven reinforcement learning,” arXiv:2004.07219, 2020.
- [45] L. Chen *et al.*, “Decision Transformer: Reinforcement learning via sequence modeling,” in *Advances in Neural Information Processing Systems*, vol. 34, 2021, pp. 15084–15097.
- [46] J. Achiam *et al.*, “Constrained policy optimization,” in *Proc. ICML*, PMLR vol. 70, 2017, pp. 22–31.
- [47] J. R. Gardner *et al.*, “Bayesian optimization with inequality constraints,” in *Proc. ICML*, PMLR vol. 32, no. 2, 2014, pp. 937–945.
- [48] J. Skalse *et al.*, “Defining and characterizing reward gaming,” in *Advances in Neural Information Processing Systems*, vol. 35, 2022, pp. 9460–9471, doi:10.52202/068431-0687.
- [49] L. Gao *et al.*, “Scaling laws for reward model overoptimization,” in *Proc. ICML*, PMLR vol. 202, 2023, pp. 10835–10866.
- [50] T. Everitt *et al.*, “Reward tampering problems and solutions in reinforcement learning: A causal influence diagram perspective,” *Synthese*, vol. 198, suppl. 27, pp. 6435–6467, 2021, doi:10.1007/s11229-021-03141-4.
- [51] R. S. Sutton *et al.*, “Between MDPs and semi-MDPs: A framework for temporal abstraction in reinforcement learning,” *Artificial Intelligence*, vol. 112, no. 1–2, pp. 181–211, 1999, doi:10.1016/S0004-3702(99)00052-1.
- [52] L. T. Clark *et al.*, “ASAP7: A 7-nm finFET predictive process design,” *Microelectronics Journal*, vol. 53, pp. 105–115, 2016, doi:10.1016/j.mejo.2016.04.006. PDK and cell libraries: <https://github.com/The-OpenROAD-Project/asap7>
- [53] D. Petrisko *et al.*, “BlackParrot: An agile open-source RISC-V multicore for accelerator SoCs,” *IEEE Micro*, vol. 40, no. 4, pp. 93–102, 2020, doi:10.1109/MM.2020.2996145.
- [54] H. Genc *et al.*, “Gemmini: Enabling systematic deep-learning architecture evaluation via full-stack integration,” in *Proc. DAC*, 2021.
- [55] Parallax Software, “OpenSTA: Static timing analyzer,” 2026. [Online]. Available: <https://github.com/The-OpenROAD-Project/OpenSTA>
- [56] YosysHQ, “EQY: Equivalence checking with Yosys,” 2026. [Online]. Available: <https://yosyshq.readthedocs.io/projects/eqy/>
- [57] C. E. Leiserson *et al.*, “Retiming synchronous circuitry,” *Algorithmica*, vol. 6, pp. 5–35, 1991, doi:10.1007/BF01759032.
- [58] M. R. Guthaus *et al.*, “OpenRAM: An open-source memory compiler,” in *Proc. ICCAD*, 2016, doi:10.1145/2966986.2980098.
- [59] R. Agarwal *et al.*, “Deep reinforcement learning at the edge of the statistical precipice,” in *Advances in Neural Information Processing Systems*, vol. 34, 2021, pp. 29304–29320.
- [60] Z. Xie *et al.*, “PowerNet: Transferable dynamic IR drop estimation via maximum convolutional neural network,” in *Proc. ASP-DAC*, 2020, pp. 13–18.
- [61] H. Wu *et al.*, “ChatEDA: A large language model powered autonomous agent for EDA,” in *Proc. MLCAD*, 2023.
- [62] U. Sharma *et al.*, “OpenROAD-Assistant: An open-source large language model for physical design tasks,” in *Proc. MLCAD*, 2024, doi:10.1145/3670474.3685960.
- [63] A. Ghose *et al.*, “Invited: Agentic AI for physical design R&D: Status and prospects,” in *Proc. ISPD*, 2026, doi:10.1145/3764386.3779612.
- [64] A. Aversa *et al.*, “Invited: 2026 Extensions of IEEE CEDA DATC’s Robust Design Flow and Other Activities,” in *Proc. ICCAD*, 2026.
- [65] M. Köfferlein, “KLayout layout viewer and editor,” 2026. [Online]. Available: <https://www.klayout.de/>
- [66] The OpenROAD Project, “Nangate45 platform and Nangate Open Cell Library,” rev. fb65d5b48. [Online]. Available: <https://github.com/The-OpenROAD-Project/OpenROAD-flow-scripts/tree/fb65d5b48/flow/platforms/nangate45>
- [67] SkyWater PDK Authors, “SKY130 high-density standard-cell library (sky130_fd_sc_hd)” [Online]. Available: https://github.com/google/skywater-pdk-libs-sky130_fd_sc_hd