Can Coding Agents Reproduce Real RTL Bugs? A CVA6 Benchmark

John Curreri

Abstract

Given a symptom-level bug report, the best coding agent we tested (Opus 5.5) writes a bare-metal test that exposes a real, recent RTL bug in the CVA6 RISC-V core in 93–98% of attempts. That rate falls to 60% when the report names only the subsystem, and to 38% when it names only the patched file. We build a benchmark of 14 functional bugs fixed upstream in CVA6 between March and September 2026; 11 of them landed in August–September, after the models' training cutoffs. The ones we use span the default, hypervisor, HPDcache and debug-trigger configurations. An agent (Claude Code, Codex CLI or Antigravity CLI, headless, in a sandbox) gets the buggy RTL, a test harness and a written report. It must produce `repro.S`, which counts only if it passes on the fixed core and does not pass on the buggy one. Over 3 samples per cell, Opus 5.5 reproduces 39/42 bugs with a simulator and 41/42 without one, and Sonnet 5.5 reproduces 39/42 and 37/42. Open-weight models served locally do far worse: GLM 5.3 Flash 20/28, Qwen3.8-27B 42/84 and Granite 4.2 8B 1/28. For Qwen, a 60-turn cap explains why the simulator seemed to hurt: with 120 turns its sim score rises from 6–7/14 to 11/14, while its no-simulator score stays at 8/14. GPT-6.1 Sol, run through Codex, reproduces 31 of the 32 episodes it is allowed to finish, but OpenAI's cyber classifier interrupts 28 of its 42 full-report episodes (94 of 126 overall), leaving it at 7–9/14 per run; Anthropic's classifiers interrupt 6% of Opus episodes and no Sonnet episode. Among models added later, Gemini 4 Argon reproduces 14/14 from full reports, Haiku 5.5 38/42 and GPT-6 Luna 33/42, while GPT-6 Astra reaches 6/14 because OpenAI's classifier interrupts most of its episodes. At API prices, Sonnet 5.5 and GPT-6.1 are the cheapest large models per reproduced bug (10–13 cents), Opus costs about twice that, and the open-weight models are no cheaper. Haiku 5.5 and GPT-6 Luna cost about one cent per reproduced bug.

1. Introduction

Turning a bug report into a failing test is the first step of fixing a hardware bug, and it is usually manual. In processor RTL, the test is a small bare-metal program. It has to drive the core into the right privilege mode, page-table state or trap path, then check an

architectural value against the specification. Writing one takes knowledge of the ISA spec, the core's microarchitecture and the test harness.

We ask whether a general coding agent can do this step alone. We pick bugs whose fixes landed upstream recently, mostly after the models' training data was collected, so the model cannot recall the fix. We grade two-sided against the real fix: a test scores only if it separates the fixed core from the buggy one.

Contributions:

1. A benchmark of 14 validated CVA6 bugs. Each has a buggy core variant, a reference reproducer, and bug reports at three levels of detail.
2. A sandboxed agent protocol with two conditions: with a simulator of the buggy core (sim) and without one (nosim). Grading runs on the host, against the pristine harness.
3. Results for two API models and three open-weight models served locally, with 3 samples per cell where budget allowed. Also an ablation over report detail, and an analysis of how agents fail.

2. Benchmark construction

The benchmark holds 14 CVA6 bugs, each confirmed by a reference test that passes on the fixed core and fails on the buggy one.

Mining. A script scans the history of CVA6, Ibex and PicoRV32 for fix commits that touch 1–2 RTL files with 40 or fewer changed lines. CVA6 gave 151 candidates, 31 of them since March 2026, against 3 for Ibex and none for PicoRV32. So CVA6 is the target core and PicoRV32 serves only to bring up the harness. Commit messages often name the faulty signal, so they are never shown to the model.

Triage and validation. A candidate is accepted only after we write a reference reproducer that fails on the buggy core and passes on the fixed one. We rejected bugs that are timing-dependent and could not be reproduced deterministically. We also rejected 07732afe5d (only visible as an X value, which two-state Verilator cannot show) and 9ada1f383c (only reachable above 2^41 physical addresses, outside the simulated memory map).

Buggy variants. Each buggy core is the CVA6 HEAD (commit 81245a47fa, 2026-09-15) with the fix reverted. Where a plain revert conflicts with later code, we wrote the reverse patch by hand (0e04a86b13, fc5fffba6f, 2d2d0c11b7). The fixed core is HEAD itself, built once per configuration with Verilator 5.008. For 49b5fa9e2f, HEAD also has a separate load-trigger hang, so its reference core is HEAD plus our own fix for that hang.

Configurations. 4 cases use the default `cv64a6_imafdc_sv39`, 7 the hypervisor config `cv64a6_imafdch_sv39`, 1 the HPDcache config, and 2 configs we added: `cv64a6_imafdc_sv39_sdtrig` (no shipped config enables debug triggers) and `cv64a6_imafdc_mu` (M and U modes only).

Case	Fixed upstream	Config	Patched RTL	Observable symptom
81245a47fa	2026-09-15	hypervisor	csr_regfile	hgatp write with an unsupported MODE is dropped entirely
d2fdf99a38	2026-09-11	default	csr_regfile	MRET/SRET to a lower mode leaves mstatus.MPRV set
49b5fa9e2f	2026-09-11	sdtrig	triggers_pkg	NAPOT execute trigger never fires
194d39face	2026-09-10	default	csr_regfile	mtval/stval not zeroed on interrupt traps
9a0e01e383	2026-09-07	hypervisor	cva6_ptw	HLVX lets the VS-stage walk read an execute-only G-stage page
6fe7b95910	2026-09-06	default	pmp_entry	PMP TOR boundary uses pmpaddr bit 0 despite 8-byte grain
99a0807757	2026-09-06	hypervisor	cva6_ptw	Bad non-leaf VS-stage PTE gives a page fault, not a guest-page fault
2d2d0c11b7	2026-09-05	hypervisor	cva6_mmu	HLV after HLVX on an execute-only G-stage page succeeds
2a12275c11	2026-09-04	hypervisor	cvxif_fu	Illegal instruction in VS-mode sets mstatus.GVA
cc81a7bb49	2026-09-04	hpdcache	csr_regfile	menvcfg.CBIE keeps the reserved encoding 2'b10

Case	Fixed upstream	Config	Patched RTL	Observable symptom
fc5fffba6f	2026-08-12	hypervisor	csr_regfile	MRET with MPP=M, MPV=1 sets V=1
18b9875e9d	2026-06-29	hypervisor	cva6_mmu, cva6_tlb	G-stage-only fetch at a GPA with bit 38 set page-faults
0e04a86b13	2026-03-17	default	decoder, branch_unit	ebreak tval is not the PC; ecall tval is not zero
aef009c4b3	2026-03-16	mu	csr_regfile, riscv_pkg	mstatus.SXL reads 64-bit on a core without S-mode

Report tiers. Each case has three reports, written by us and never copied from the commit. The *full* report describes the symptom in a paragraph. The *vague* report names the subsystem in one sentence, e.g. "MRET does not update mstatus correctly in some cases." The *loc* report names only the patched files and says the core deviates from the spec there.

3. Evaluation setup

Every model runs as the same agent, headless Claude Code 2.1.302, in a locked-down Docker container. It is scored only on whether its test separates the fixed core from the buggy one.

Workspace. The container (`rtlbench-agent:2.1.302`) has no git, curl, wget or Python, and mounts only the episode workspace. The workspace holds the buggy `core/` RTL with no git history, the harness (`test.h` , `hyp.h` with two-stage page-table helpers, linker script, Makefile), a RISC-V GCC toolchain and `TASK.md` . `TASK.md` contains the report and the rules, under an anonymized case id. The agent must leave a `repro.S` that ends in `PASS` on a correct core and does not end in `PASS` on the buggy core; a FAIL, an unexpected trap or a hang all count.

Conditions. In *sim*, the workspace also holds a Verilator build of the buggy core and `sim/run.sh` . That script builds and runs a test in about 15 s, printing SUCCESS, FAILED or TIMEOUT plus an instruction trace. In *nosim*, the agent can compile but not run anything. Neither condition ever gives it the fixed core.

Grading. On the host, `grade.py` rebuilds `repro.S` against the pristine harness, not the agent's copy. It then runs the test on the fixed and the buggy core, each in a fresh directory with a 2M-cycle limit. An episode scores 1 only if the fixed core passes and the buggy core does not.

Models and backends.

Model	Served by	Context	Episodes per cell
Claude Opus 5.5	Anthropic API (subscription)	default	3
Claude Sonnet 5.5	Anthropic API (subscription)	default	3
GLM 5.3 Flash	FreeToken, local, Anthropic-compatible API	131k	1
Qwen3.8-27B (NVFP4)	SGLang, RTX PRO 6000	262k	3
GPT-6.1 Sol (effort high; one run at xhigh)	OpenAI via Codex CLI 0.163.0 (subscription)	default	3
Granite 4.2 8B (BF16)	SGLang, RTX PRO 6000	131k	1
Claude Haiku 5.5	Anthropic API (subscription)	default	3
GPT-6 Astra (effort high)	OpenAI via Codex CLI (subscription)	default	1
GPT-6 Luna (effort high)	OpenAI via Codex CLI (subscription)	default	3
Gemini 3.8 Flash (high)	Google via Antigravity CLI (subscription)	default	1
Gemini 4 Argon (high)	Google via Antigravity CLI (subscription)	default	1 (full tier only)

Local models get no Anthropic credentials. Every episode has a 60-turn cap, except the Qwen3.8-27B runs marked 120-turn, which test the cap's effect: one full-tier run per condition, plus one vague and one loc run. Qwen's chat template rejects Claude Code's default reasoning effort `high`, so it runs with `xhigh`; an earlier Qwen run with the default (`qwen1`, all HTTP 500) is excluded.

Void episodes. An episode cut off by the subscription usage limit, or by an API error from the provider, is void rather than a miss. It is deleted and rerun. Final API-model cells contain no void episodes. Local-model API errors (Granite exceeding its 131k context) count as misses, because they are the model's own failure. Sim runs made between 2026-10-20 and 2026-10-22 had a simulator that could not start inside the container (a missing library path); they were discarded and rerun with the fix, except 6 Granite sim episodes (Section 7).

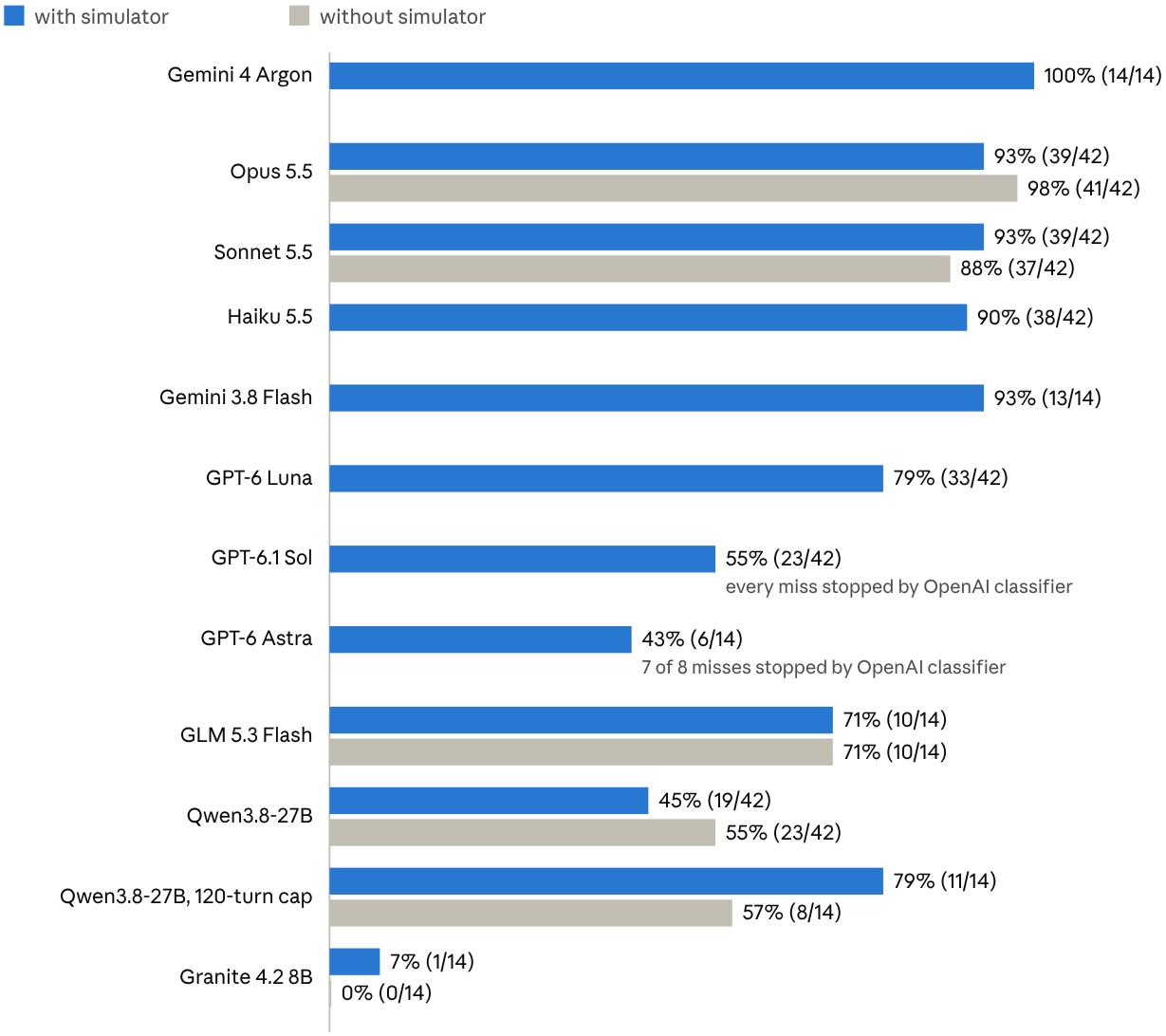
Sandbox-escape flag. The runner flags any tool call that mentions a URL, git, curl, wget, the upstream org or the benchmark's case directory. Two Sonnet episodes were flagged, both for calling git, which the container does not have; one of them scored. Among local models, 18 episodes were flagged, and 5 of those scored. The container needs the network for the model API, but has no download tools and no mount of the case directory, so a flag marks an attempt rather than a leak.

4. Results

With a full report, both Claude models reproduce nearly every bug, and the simulator makes little difference to them. The open-weight models are far behind, and the smallest, Granite 4.2 8B, almost never succeeds. GPT-6.1 reproduces every full-report bug its provider's classifier lets it finish, 23 of 42 (Section 5). Of the models added in October, Gemini 4 Argon reproduces all 14, and the small hosted models Haiku 5.5, Gemini 3.8 Flash and GPT-6 Luna reach 79–93%.

Full reports: Gemini Argon 100%, Claude 88–98%, Luna and Flash 79–93%, open-weight 0–79%

Share of episodes whose test passes on the fixed core and fails on the buggy core (full report)



GPT-6.1, Astra, Luna: Codex CLI, effort high. Argon, Gemini Flash: Antigravity CLI. Models added in October ran with the simulator only. n = 42 is 3 samples of 14 cases; n = 14 is 1 sample.

eval/runs/*/result.json · Opus, Sonnet, Qwen, Haiku, Luna: 3 samples × 14 cases; GLM, Granite, Qwen 120-turn, Argon, Gemini Flash, Astra: 1 sample; GPT-6.1: codex-full-s1–s3 (sim, effort high)

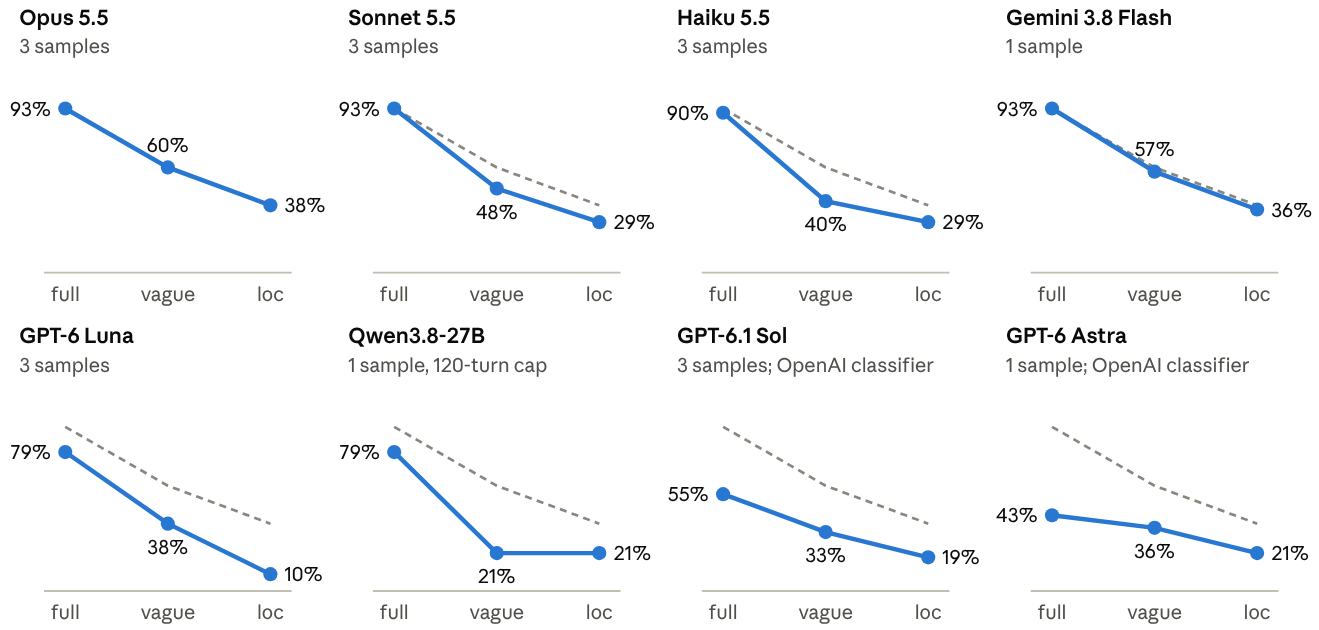
Across samples, Opus scored 12–14 of 14 per run in each condition, and Sonnet 11–14. The two Claude models are within noise of each other: 80/84 against 76/84 episodes. GLM 5.3 Flash scores the same with and without the simulator. Qwen3.8-27B scores lower with it at a 60-turn cap, and higher with it once the cap is doubled (Section 5).

Report detail. How much the report says matters far more than the simulator. A one-sentence subsystem hint costs Opus over a third of its solves (39 to 25 of 42), and a bare file name costs it nearly 60% (16 of 42). Sonnet, level with Opus on full reports, falls further: to 48% and then 29%. Qwen3.8-27B at a 120-turn cap falls from 11 of 14 to 3 of 14

at both shorter tiers (1 sample each), and GPT-6.1 from 23 to 14 to 8 of 42, as its classifier stops more episodes when the report says less. The small hosted models fall as far or further: Haiku 5.5 from 90% to 29%, Gemini 3.8 Flash to 36% and GPT-6 Luna to 10%. With only the file name, the agent must also find the right bug: a test that exposes a different real bug in the same file fails on both cores and scores zero.

Shorter reports cut every model: Opus falls from 93% to 38%, the others to 10–36%

Bugs reproduced with the simulator, by report tier
full = whole report · vague = one subsystem sentence · loc = patched file name only · dashed grey = Opus 5.5, for reference



eval/runs/{opus,sonnet,haiku,luna}-{full,vague,loc}-s*/, opus1, batch1, qwen-turns120, qwen120-{vague,loc}-s1, codex-{full,vague,loc}-s1–s3, codex-astra-*-s1, gemini-flash-*-s1 · sim condition; Argon ran the full tier only

Per-case difficulty. No case defeats Opus or Sonnet; each solved every case in at least 4 of its 6 episodes, and Argon solved all 14. The smaller models have blind spots the larger ones do not: Haiku never solved 0e04a86b13 and Luna never solved 49b5fa9e2f, though Opus and Sonnet solved both in 11 or more of 12 episodes. The hardest cases overall are the hypervisor MMU bugs (9a0e01e383, 2d2d0c11b7, 18b9875e9d), which need two-stage page tables set up correctly before the bug is even reachable. The easiest is aef009c4b3, a single CSR read on an M/U-only core.

Case	Opus (of 6)	Sonnet (of 6)	Haiku (of 3)	Argon (of 1)	Gemini Flash (of 1)	GPT-6.1 (of 3)	Luna (of 3)	Astra (of 1)	GLM (of 2)	Qwen (of 6)	Granite (of 2)	Total (of 34)
aef009c4b3	6	6	3	1	1	3	3	1	2	5	1	32

Case	Opus (of 6)	Sonnet (of 6)	Haiku (of 3)	Argon (of 1)	Gemini Flash (of 1)	GPT-6.1 (of 3)	Luna (of 3)	Astra (of 1)	GLM (of 2)	Qwen (of 6)	Granite (of 2)	Total (of 34)
81245a47fa	6	6	3	1	1	3	3	1	2	4	0	30
2a12275c11	6	6	3	1	1	2	3	0	2	5	0	29
cc81a7bb49	6	5	3	1	1	3	1	1	2	5	0	28
194d39face	6	6	3	1	1	3	3	1	1	3	0	28
d2fdf99a38	5	6	3	1	1	2	3	0	1	4	0	26
fc5fffba6f	6	4	3	1	1	1	3	0	1	5	0	25
99a0807757	6	5	3	1	1	0	3	1	1	2	0	23
49b5fa9e2f	6	5	3	1	1	3	0	0	1	2	0	22
6fe7b95910	6	6	3	1	1	0	2	0	1	2	0	22
0e04a86b13	6	6	0	1	0	3	2	1	1	1	0	21
18b9875e9d	6	4	2	1	1	0	3	0	1	2	0	20
2d2d0c11b7	5	6	3	1	1	0	2	0	2	0	0	20
9a0e01e383	4	5	3	1	1	0	2	0	2	2	0	20

Counts are bugs reproduced over the sim and nosim episodes of the full-report tier; GPT-6.1 has three sim runs at effort high. Its five zeros are all cases OpenAI's classifier stopped on every run. The models added in October have sim runs only: three for Haiku and Luna, one for Argon, Gemini Flash and Astra. All three of Opus's sim misses (two on 9a0e01e383, one on 2d2d0c11b7) are episodes a safety classifier stopped within 5 turns; a retry of the first scored, but the original stays as the sample. Five of Opus's six solves of 6fe7b95910 and one of 9a0e01e383 ran mostly on Opus 4.8 (Section 7).

Models added 25–26 October

Gemini 4 Argon reproduced all 14 bugs from full reports, the first perfect run in the benchmark (Opus and Sonnet: 39/42 each). The three small models, Haiku 5.5, GPT-6 Luna and Gemini 3.8 Flash, come close to that with full reports but fall off sharply with vague or localization-only reports. Argon's vague and loc cells were not run: the Antigravity weekly quota ran out after its full-report run and resets after our deadline.

Model	Samples	Full	Vague	Loc
Gemini 4 Argon (high)	1	14/14	not run (weekly quota)	not run (weekly quota)

Model	Samples	Full	Vague	Loc
Gemini 3.8 Flash (high)	1	13/14	8/14	5/14
Claude Haiku 5.5	3	38/42	17/42	12/42
GPT-6 Luna (high)	3	33/42	16/42	4/42
GPT-6 Astra (high)	1	6/14	5/14	3/14

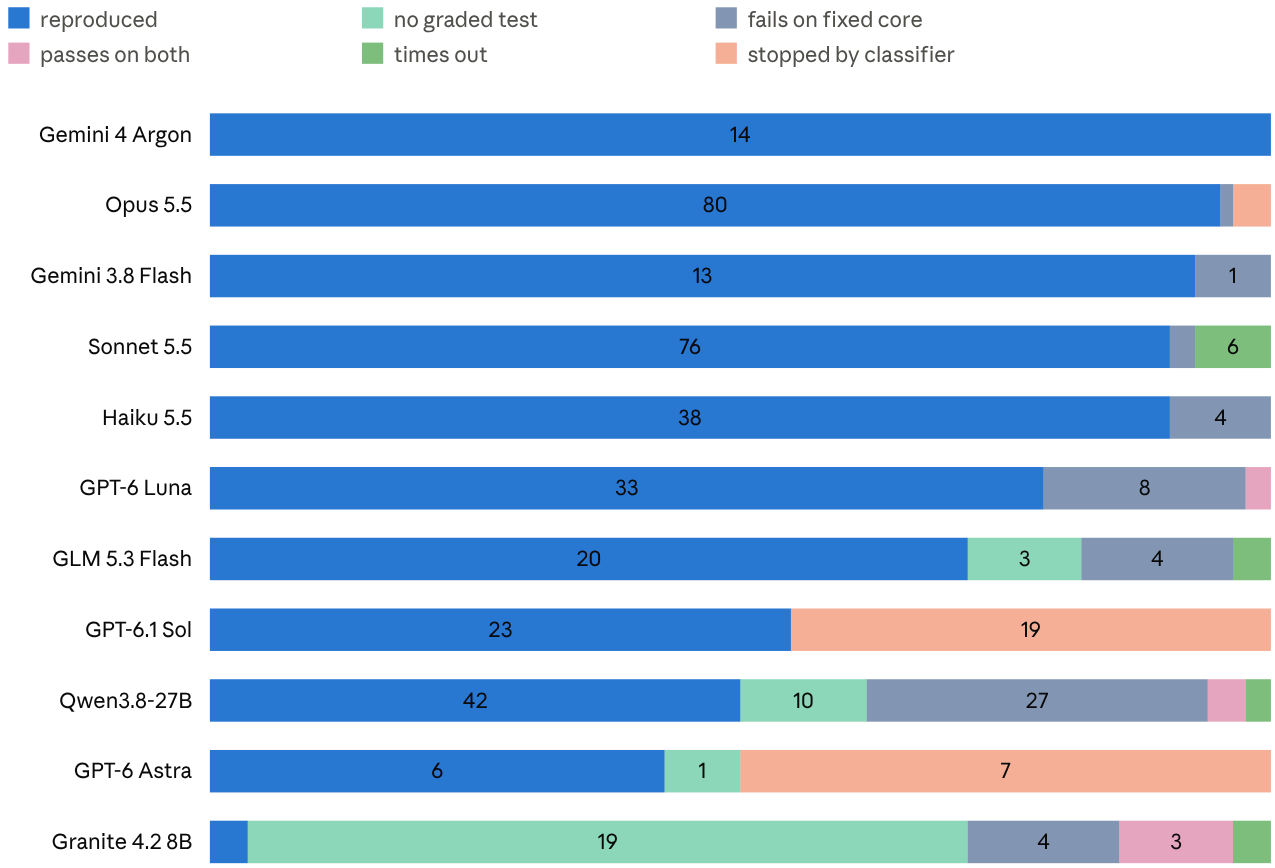
Sim condition; run ids `gemini-argon-*`, `gemini-flash-*`, `haiku-*`, `luna-*`, `codex-astra-*` in `eval/runs`, as of 26 October 10:15 CDT. Episodes voided by a subscription quota ("usage limit", "Individual quota reached") were deleted and rerun, never counted as failures. Astra scores below Sol because OpenAI's classifier interrupted 30 of its 42 episodes (Section 5). Luna's loc misses are ordinary: 24 tests fail on both cores and 4 time out. Two Flash episodes (one vague, one loc) ended on a malformed function call from the model and count as failures.

5. Analysis

The models fail in different ways: Claude models rarely fail at all, GPT-6.1 and GPT-6 Astra fail almost only when OpenAI's classifier ends the session, Qwen writes tests that reject the correct core, and Granite usually never produces a test that builds. The small hosted models fail the way Qwen does, but far less often: every Haiku and Gemini Flash miss, and 8 of Luna's 9, is a test that fails on the fixed core.

Qwen writes wrong tests, Granite none that build, both GPT-6 models get stopped

How each full-report episode ended: sim and nosim pooled, added models sim only (share of episodes; counts on the bars)



GPT-6.1: 3 runs at effort high. Another 6 Opus, 9 GPT-6.1 and 1 Astra episodes hit a classifier but still reproduced.
Astra's one no-test episode is the model declining the task. Luna's fixed-core failures include 3 tests that hang on the fixed core.

eval/runs/*/result.json grade fields · full report tier; 'no graded test' = no repro.S, or one that does not build; added models (Haiku, Argon, Gemini Flash, Luna, Astra) sim only

Wrong tests. A test that fails on the fixed core encodes a wrong reading of the spec, or sets up state the core does not support. This is Qwen's main failure (27 of 84 episodes). The simulator cannot catch it, because the agent only ever sees the buggy core: a FAIL there looks like success.

No test. Granite 4.2 8B wrote a `repro.S` that did not assemble in 13 of 28 episodes, and none at all in 6. In those episodes it did invoke the build, often 5–13 times, but stopped with an empty final message while the file still failed to assemble. Two Granite episodes ended when the conversation overflowed its 131k context; in one, Claude Code's automatic compaction also failed.

Timeouts. Sonnet's 6 non-reproductions are all tests that hang on at least one core within 2M cycles. Five of the six are hypervisor cases, where the test must build two-stage page tables and trap handlers before reaching the bug.

The turn cap explains why the simulator seemed to hurt Qwen. With the simulator, 8–10 of 14 Qwen episodes per run hit the 60-turn cap while still iterating, and it reproduced 6–7 bugs. Without it, Qwen stops earlier (median 42 turns against 61) and scores 7–9. With a 120-turn cap and the simulator, it reproduces 11 of 14. Without the simulator, the higher cap changes nothing (8 of 14; no episode passed 68 turns). Given enough turns, the simulator helps Qwen by 3 bugs. For the Claude models, the median episode takes 10 turns (Opus) and 8 (Sonnet), so the cap never binds with full reports. With shorter reports it does: 10 of Opus′s 17 vague-tier misses and 10 of its 26 loc-tier misses ran out of turns, as did 5 of Sonnet′s 22 vague misses and 10 of its 30 loc misses.

Shorter reports. With only the patched file named, Opus′s most common failure (20 of 42 episodes) is a test that fails on both cores. Such a test either targets a different behaviour in the file than the one patched, or misreads what the spec requires; a hand classification of 40 such Opus and Sonnet tests from an earlier set of loc runs, since repeated, found 24 that expose real deviations still in the fixed core, 9 spec misreadings and 7 broken tests. With the subsystem sentence, the same failure falls to 7 of 42, and running out of turns is Opus′s most common failure (10 of 42).

Safety classifiers

OpenAI′s safety classifier, not the model, sets GPT-6.1′s score. All but one GPT-6.1 miss was a session the classifier interrupted; of the 32 episodes it let run to the end, GPT-6.1 reproduced 31.

We ran GPT-6.1 Sol through Codex CLI 0.163.0 on a ChatGPT subscription, in the same sandbox and with the simulator, at reasoning effort high, the same effort Claude Code uses by default for Opus and Sonnet. Codex has no turn cap, but no episode used more than 24 tool calls. When the classifier fires, the session ends with "This content was flagged for possible cybersecurity risk" and no fallback.

Report	GPT-6.1 reproduced (per run)	GPT-6.1 interrupted by classifier	GPT-6 Astra reproduced (1 run)	Opus 5.5 reproduced	Sonnet 5.5 reproduced
full	23/42 (7, 7, 9)	28 (9 had already written a passing test)	6/14 (8 interrupted)	39/42	39/42
vague	14/42 (4, 4, 6)	30 (2 still scored)	5/14 (10 interrupted)	25/42	20/42
loc	8/42 (2, 2, 4)	36 (3 still scored)	3/14 (12 interrupted)	16/42	12/42

All three models: 3 samples per tier, with the simulator. The classifier fires more as the report gets vaguer (28, 30 and 36 of 42 episodes), perhaps because the agent then probes privilege and memory-protection behaviour more widely. The one GPT-6.1 miss it did not cause is a loc-tier test of 194d39face that fails on both cores.

Repeatability. The three full-tier runs scored 7, 7 and 9 of 14, and a fourth run at effort xhigh scored 6, so effort does not change the outcome. Across the three runs at high, 8 cases were flagged every time, 3 never, and 3 varied. In all, OpenAI's classifier interrupted 94 of 126 GPT-6.1 episodes (75%).

Anthropic's classifiers fired far less often. Across all 198 Opus 5.5 episodes, including reruns, 12 had a classifier event, all on three cases (6fe7b95910, 9a0e01e383, 2d2d0c11b7) and none at the loc tier. In 8, the session switched to Opus 4.8 and continued; in 4, a response was blocked and the episode ended early. No setting of these cases triggered on every run except 6fe7b95910 at full/nosim (3 of 3). Sonnet 5.5 had no classifier event in 225 episodes. Each rerun writes different code and text, so we cannot tell classifier randomness from trajectory differences; what is clear is that the chance of a trigger depends strongly on the case.

Added models. GPT-6 Astra drew OpenAI's classifier on 30 of its 42 episodes (8 full, 10 vague, 12 loc), more often than Sol at every tier. In one further full-report episode (2d2d0c11b7) the model itself declined after reading the task: "I can't provide code that reproduces that isolation failure". That is the only model-level refusal in the benchmark. On the same account and tasks, GPT-6 Luna had no classifier event in 126 episodes. Haiku 5.5 had none in 126 episodes, and Gemini 3.8 Flash and Gemini 4 Argon had none.

Model or provider? We ran Opus 5.5 (high) through Google's Antigravity CLI on the three cases where Anthropic's classifier fired under Claude Code, 3 samples each. Antigravity's content-safety filter ended 7 of the 9 episodes after 3–19 turns. Six of those had already

written a `repro.S` that grades valid, so the filter cut the session short without costing the result; the seventh (6fe7b95910, third sample) was blocked at turn 3, before any test existed. Only 2d2d0c11b7 ran to the end unblocked, in the second and third samples. In all, 8 of 9 episodes reproduced the bug. Under both providers, then, Opus on these cases draws a safety filter, which points to the task content rather than to one provider's classifier.

On security-adjacent hardware verification, how a vendor's classifier handles a false positive changes the score more than the gap between the models does.

Cost per reproduced bug

At API list prices, Sonnet 5.5 (10–13 cents) and GPT-6.1 (11 cents) reproduce a bug most cheaply; Opus 5.5 costs about 2× that, and the open-weight models are not cheaper per bug.

Model	Condition	Reproduced	$ per episode	$ per bug	$ per bug, no caching
GPT-6 Luna (high)	sim	33/42	0.01	0.012	–
Haiku 5.5	sim	38/42	0.013	0.014	–
Sonnet 5.5	nosim	37/42	0.08	0.10	0.10
Sonnet 5.5	sim	39/42	0.12	0.13	0.13
GPT-6.1 Sol (high)	sim	23/42	0.08	0.11	0.11
GLM 5.3 Flash	sim, nosim	10/14 each	0.13	0.18	0.59
GPT-6.1 Sol (xhigh)	sim	6/14	0.10	0.18	0.18
Opus 5.5	sim	39/42	0.25	0.27	0.27
Opus 5.5	nosim	41/42	0.28	0.29	0.29
GPT-6 Astra (high)	sim	6/14	0.31	0.72	–
Gemini 3.8 Flash (high)	sim	13/14	0.75	0.81	3.96
Qwen3.8-27B, 120-turn cap	sim	11/14	0.73	0.93	3.43
Qwen3.8-27B	nosim	23/42	0.51	0.94	3.32
Gemini 4 Argon (high)	sim	14/14	1.03	1.03	5.92
Qwen3.8-27B	sim	19/42	0.70	1.54	5.76

Haiku 5.5's cost is the API-equivalent figure Claude Code reports. Astra and Luna are priced from Codex token counts at OpenAI Standard list prices per 1M input / cached / output tokens: Astra $10 / $1.00 / $50, Luna $0.10 / $0.01 / $0.50. We did not compute the

no-caching column for these three. The Gemini models are priced from Antigravity's token counts at Google's paid-tier prices per 1M input / cached / output tokens, thinking billed as output: Flash $0.75 / $0.075 / $3.75 (through 31 December 2026), Argon $2.00 / $0.10 / $10.00 (through 30 November 2026). Both double after those dates. Cache storage fees are left out. Their cost comes mostly from cached context: a Flash episode reads 4.3M cached tokens on average over about 76 tool steps.

Full-report tier. Costs are recomputed from each episode's token counts at list prices per million tokens, input / cache read / output: Opus 5.5 $4 / $0.20 / $20 (cache writes $8, the 1-hour rate Claude Code uses), Sonnet 5.5 $2 / $0.20 / $10 (writes $4), GPT-6.1 Sol $2 / $0.10 / $10, GLM-5.3-Flash on Z.ai $0.15 / $0.03 / $0.50, and Qwen3.8-27B on Alibaba Model Studio $0.50 / $0.05 / $3.00. For Opus this reproduces Claude Code's own cost figure exactly.

Two estimates. Our local servers did no prompt caching, so a GLM or Qwen episode re-sends its whole context every turn, about 2.5M input tokens. The "$ per bug" column assumes an API would serve 93.8% of input from cache, the rate Opus measured; the last column prices every input token in full. GPT-6.1 episodes that the classifier stopped report no usage, so each is charged at two-thirds of a clean episode, in proportion to their tool calls (median 7 against 11).

Why open weights are not cheaper. GLM and Qwen take about 56–58 turns per episode against about 10 for Claude, and succeed less often. Qwen is the most expensive per bug: Alibaba lists its output at $3, and it reproduces only half the bugs. Third-party hosts list Qwen3.8-27B at about $0.02–0.05 input and $0.28–0.90 output, which would cut its cost per bug by roughly 5–10×. These are API-equivalent figures; all our runs used subscriptions or local GPUs.

6. Related Work

Prior hardware benchmarks ask a model for a design or a patch; prior bug-reproduction benchmarks ask for a test, but only for software. This benchmark asks for a test that reproduces a real bug in processor RTL.

Work	Model produces	Bugs	Oracle
VerilogEval [2], RTLLM [3]	A Verilog module from a spec	None (generation)	Simulation against a reference design

Work	Model produces	Bugs	Oracle
CVDP [4]	Designs, testbenches, fixes, answers (13 categories)	Problems written by hardware engineers	Per-task harness
Fixbench-RTL [5]	A fix for buggy RTL	Synthetic and mined from version history	LLM-built testbench: fails on buggy, passes on fixed
HWE-Bench [6]	A repository patch	417 real fix PRs in 6 projects, 35 in CVA6	LLM-written test script per task
LIBRO [7], SWT-Bench [8]	A test reproducing a bug report	Real Java and Python issues	Fails before the fix, passes after
ChatFuzz [9]	Instruction streams	Unknown bugs (fuzzing)	Mismatch against an ISA golden model

RTL generation and repair. VerilogEval [2] and RTLLM [3] score Verilog written from a specification. CVDP [4] widens this to 783 problems, including verification and debugging, in single-turn and agentic formats; state-of-the-art models reach at most 34% pass@1 on its generation tasks. Fixbench-RTL [5] and HWE-Bench [6] ask for repairs. HWE-Bench is the closest in setting: real fix PRs from CVA6 and five other projects, run by agents in containers. Its best agent resolves 70.7% of tasks. Both use the same fail-on-buggy, pass-on-fixed rule as ours, but there an LLM writes the test while building the benchmark; it is not what the model under test is scored on. HWE-Bench estimates cost per task from API prices (Sonnet 4.6 $0.26, Opus 4.6 $0.51, GPT-5.4 xhigh $1.81), but not cost per resolved task, and it does not report safety-classifier interventions or their effect on scores, although two of its six projects are security roots of trust. Its models are also a generation older: GPT-5.4 at xhigh effort against Opus and Sonnet 4.6 at high.

Bug reproduction in software. LIBRO [7] first had LLMs write tests from bug reports, reproducing 33% of 750 Defects4J bugs. SWT-Bench [8] turned this into a benchmark for code agents on real GitHub issues and found that repair agents beat purpose-built test generators. Our task is the hardware analogue, and our grading rule is theirs, applied to two simulated cores instead of two versions of a program.

LLM-guided processor testing. ChatFuzz [9] uses an LLM to generate instruction streams for a RISC-V core and finds bugs by mismatch with a golden model. It hunts for unknown bugs by coverage; it does not start from a report of a known one.

What is new here.

- **The test is the output.** The agent must turn a written report into a program that exposes the bug. No hardware benchmark we know of scores that.
- **A real application-class core.** All 14 bugs are real upstream CVA6 fixes, including hypervisor and debug-trigger cases that need two-stage page tables and trap handlers before the bug can be reached.
- **A two-sided oracle on real RTL.** Each test runs on both the fixed and the buggy core in cycle-accurate Verilator, so a test that also fails on the correct core does not count.
- **Controlled variables.** Report detail (full, vague, loc), simulator access (sim, nosim) and turn cap are varied independently, and 11 of the 14 fixes postdate the models' training data.
- **Real findings as a by-product.** At the loc tier, many failing tests hit real deviations still present in the fixed core, one of which was reported to the maintainers.
- **Classifier impact and cost per bug.** We measure how often each vendor's safety classifier interrupts the agent, whether it fires again on a rerun, and how its handling changes the score, and we price every model per reproduced bug rather than per attempt (Section 5). No hardware-agent benchmark we know of reports classifier interventions.

7. Limitations

The results are from 14 bugs in one core, so per-model differences of a few episodes are within noise.

- **Small benchmark.** 14 cases, 10 of them in the CSR file or the MMU. Opus and Sonnet are at 88–98% on the full tier, so that tier no longer separates them.
- **Contamination.** Three fixes (0e04a86b13, aef009c4b3, 18b9875e9d) landed in March and June 2026 and may be in the models' training data. aef009c4b3 is the easiest case overall, while the other two rank among the hardest, so there is no consistent sign of recall.
- **Uneven sampling.** GLM, Granite and the Qwen 120-turn runs have 1 sample per cell. Single runs are noisy: one Opus vague-tier sample scored 11 of 14 with the broken simulator and 7 of 14 when rerun.
- **Loc-tier undercount.** A test that exposes a different real bug in the same file scores zero, so the loc tier measures finding *the* bug, not *a* bug.

- **Altered reference core.** 49b5fa9e2f is graded against HEAD plus our own fix for a separate trigger-load hang, not against upstream HEAD.
- **Same agent, different models.** Every model except GPT-6.1 runs inside Claude Code, which was built and tuned for Claude models. Open-weight models may do better in an agent designed for them.
- **Model fallback.** Opus episodes are labeled by the requested model. In 8 of them (7 scored), a safety classifier (category "cyber") stopped a response and Claude Code served the rest of the session with Opus 4.8; 6 of the 8 are on the PMP case 6fe7b95910. In 3 more, the classifier stopped the episode within 5 turns with no fallback. Excluding the 8, Opus scores 37/40 (sim) and 37/38 (nosim) on the full tier, and 24/40 on the vague tier.
- **GPT-6.1 coverage.** GPT-6.1 has 3 samples per tier, with the simulator only, and runs in Codex CLI rather than Claude Code. Codex has no turn cap; no episode needed more than 24 tool calls. Codex does not report which model served each request, so episodes are labeled by the model requested.
- **Estimated cost.** Costs are API-equivalent at October 2026 list prices; our runs used subscriptions and local GPUs. The local servers did no prompt caching, so GLM and Qwen costs assume Opus's 93.8% cache-hit rate. GPT-6.1 episodes stopped by the classifier report no usage and are estimated from their tool calls.

8. Conclusion

Given a clear symptom report, a general coding agent with a frontier model can turn a real, recent CVA6 bug into a two-sided failing test almost every time, without simulator access. The hard part of the task is localization: as the report shrinks to a subsystem or a file name, Opus's rate falls from 93% to 60% to 38%, and Sonnet's from 93% to 48% to 29%. Open-weight models up to 27B parameters stay well below the Claude models, and their failures are basic: wrong readings of the spec, tests that do not assemble, and running out of turns. Two practical results matter as much as the model ranking. First, safety classifiers decide some scores: OpenAI's ended most GPT-6.1 episodes on hypervisor and memory-protection bugs, though the model reproduced 31 of the 32 episodes it was allowed to finish, while Anthropic's fired on 6% of Opus episodes and usually let the session continue on Opus 4.8. Second, cost per reproduced bug favors Sonnet 5.5 and GPT-6.1: the open-weight models need about five times as many turns and fail more often, so their low token prices do not make them cheaper. Small hosted models change that

picture: with full reports, Haiku 5.5 and GPT-6 Luna reproduce 38/42 and 33/42 bugs at about one cent each, though both fall well behind with vague or localization-only reports. Gemini 4 Argon is the only model to reproduce all 14 bugs from full reports in one sample.

Next steps:

1. More cores and bug classes, especially timing-dependent bugs (the five triaged-out MMU and LSU races) and bugs outside the CSR file and MMU.
2. Repeat GLM and Granite to 3 samples.
3. Classify the current loc-tier tests that fail on both cores (wrong bug, or wrong spec reading); only an earlier set is classified so far.
4. Run the open-weight models with a 120-turn cap and a build check before the agent may stop.
5. Give the agent the 7 cases OpenAI's classifier always stops as repair tasks (same bug and report, write a patch, graded by our reference test). If those run clean, the classifier is reacting to exploit-shaped tests, not to hardware security as a topic.
6. Rerun GPT-6.1 with OpenAI's trusted cyber access, if granted, to separate the model from its classifier.

Artifacts. The benchmark scripts, harness, case definitions (buggy-variant patches, reports, reference tests) and per-episode results with grades and transcripts are released at github.com/j-curreri/rtl-bug-repro-bench (reference 1). CVA6 itself is fetched from upstream at commit 81245a47fa. Four episodes are withheld in part: in each, the agent found a separate, unfixed deviation in CVA6 that has been reported privately to its maintainers. Their tests and transcripts stay out of the release until it is fixed; their grades are kept in the results.

References

1. rtl-bug-repro-bench: benchmark code, harness, cases and per-episode results; agent transcripts are attached to the GitHub release.
2. M. Liu, N. Pinckney, B. Khailany, H. Ren. VerilogEval: Evaluating Large Language Models for Verilog Code Generation. ICCAD 2023.
3. Y. Lu, S. Liu, Q. Zhang, Z. Xie. RTLLM: An Open-Source Benchmark for Design RTL Generation with Large Language Model. ASP-DAC 2024.
4. N. Pinckney, C. Deng, C.-T. Ho, Y.-D. Tsai, M. Liu, W. Zhou, B. Khailany, H. Ren. Comprehensive Verilog Design Problems: A Next-Generation Benchmark Dataset for Evaluating Large Language Models and Agents on RTL Design and Verification.

arXiv:2506.14074, 2025.

5. S. Li, W. Fu, Y. Zhao, X. Guo, Y. Jin. Fixbench-RTL: A Comprehensive Benchmark for Evaluating LLMs on RTL Debugging. AsianHOST 2025.
6. F. Cui, H. Hou, Z. Luo, C. Yin, Y. Liang. HWE-Bench: Benchmarking LLM Agents on Real-World Hardware Bug Repair Tasks. arXiv:2604.14709, 2026.
7. S. Kang, J. Yoon, S. Yoo. Large Language Models are Few-shot Testers: Exploring LLM-based General Bug Reproduction. ICSE 2023.
8. N. Mündler, M. N. Müller, J. He, M. Vechev. SWT-Bench: Testing and Validating Real-World Bug-Fixes with Code Agents. arXiv:2406.12952, 2024.
9. M. Rostami, M. Chilese, S. Zeitouni, R. Kande, J. Rajendran, A.-R. Sadeghi. Beyond Random Inputs: A Novel ML-Based Hardware Fuzzing (ChatFuzz). arXiv:2404.06856, 2024.https://claude.ai/artifact/2Bjbu8hPT2SSmthMa5kme5#ce0a46da-298b.m37chpsxpve.10938