Did Opus 5 study for the test? — full task detail

Every task in both arms: language, upstream source, merge date, grading coverage and outcome.

Two 13-task arms were run back-to-back against Claude Opus 5 at medium effort, one attempt each, judges disabled. They differ in a single property: whether the upstream fix was published **before** the model's May 2026 training cutoff (so it could have been memorised) or **after** it (so it could not). Every task is a real merged pull request, graded by hidden tests in a network-isolated Docker sandbox, and pre-validated so the gold patch scores 1.0, the unpatched repo scores 0.0, and grading is deterministic over three runs.

Result

Could have seen it (pre-cutoff)	**11 / 13** — 1 wrong answer, 1 ran out of time
Never saw it (post-cutoff)	**12 / 13** — 1 wrong answer
Genuine capability misses	**1 versus 1**
Stalls / infrastructure errors	0 / 0 on both arms
Spend	$16.36 vs $4.58 — $20.94 total

Reading the gap. The visible one-task difference is inside the noise floor at n=13, where a single task is 7.7 pp. The arms are matched on language but *not* on repo size, and the only non-capability failure landed on the largest repo in the study. Cost tracks the same asymmetry: the pre-cutoff arm carries three more large-or-xlarge repos and cost 3.6× as much to run.

Composition — the arms are language-matched, not size-matched

Language	Could have seen it	Never saw it
python	3	3
rust	4	4
typescript	3	3
javascript	1	1
go	2	2
total	**13**	**13**

Repo scale	Could have seen it	Never saw it
medium	2	5
large	3	5
xlarge	8	3
large + xlarge	**11**	**8**

Why this matters. Language is perfectly matched across the arms — 3 Python, 4 Rust, 3 TypeScript, 1 JavaScript, 2 Go on each side — so language cannot explain the difference. Repo scale is not matched, and per-step cost rises with repo size, so a scale-heavy arm loses more runs to the clock and costs more. Report scale composition beside any cross-suite score.

Could have seen it — fixes merged 2026-02-12 to 2026-05-21 (before the cutoff)

Task	Lang	Upstream repo	PR	Merged	Scale	Category	Difficulty	f2p	p2p	Sec	Steps	Cost	Outcome
flask-teardown-robust	python	pallets/flask	#5928	2026-02-19	medium	bug fix	hard	3	1	432	200	$1.55	✓
networkx-leiden-communities	python	networkx/networkx	#8509	2026-05-12	xlarge	feature	hard	6	1	3606	274	$7.68	✗ ran out of time
sqlglot-canonicalize-internal-names	python	tobymao/sqlglot	#7580	2026-05-04	large	feature	hard	6	1	1744	298	$4.90	✗ wrong answer
itertools-strip-prefix	rust	rust-itertools/itertools	#1104	2026-05-21	xlarge	feature	medium	4	1	316	104	$0.42	✓
jiff-signdur-panic	rust	BurntSushi/jiff	#527	2026-02-28	xlarge	bug fix	medium	3	3	60	58	$0.12	✓
jiff-date-day-lt1	rust	BurntSushi/jiff	#524	2026-02-22	xlarge	bug fix	medium	3	3	81	74	$0.13	✓
jiff-strftime-negpad	rust	BurntSushi/jiff	#487	2026-02-12	xlarge	bug fix	medium	3	3	193	142	$0.51	✓
zod-invert-codec	typescript	colinhacks/zod	#5770	2026-04-28	xlarge	feature	medium	5	1	230	130	$0.53	✓
zod-proto-catchall	typescript	colinhacks/zod	#5898	2026-04-29	xlarge	bug fix	medium	3	1	61	62	$0.14	✓
hono-request-bytes	typescript	honojs/hono	#4921	2026-05-16	xlarge	feature	medium	4	1	115	42	$0.07	✓
semver-truncate	javascript	npm/node-semver	#855	2026-05-07	medium	feature	medium	4	1	178	52	$0.12	✓
chi-readfrom-tee-doublecount	go	go-chi/chi	#1085	2026-05-16	large	bug fix	medium	3	1	185	42	$0.11	✓
cobra-noduplicateargs	go	spf13/cobra	#2397	2026-04-25	large	feature	medium	3	1	96	42	$0.07	✓

f2p = fail_to_pass tests, each verified to fail at the base commit and pass after the gold patch. **p2p** = pass_to_pass regression guards, which compile and pass at the base commit. Shaded rows are the tasks that failed. **Sec** and **Steps** are the graded run's wall-clock and agent steps.

Never saw it — fixes merged 2026-06-10 to 2026-07-19 (after the cutoff)

Task	Lang	Upstream repo	PR	Merged	Scale	Category	Difficulty	f2p	p2p	Sec	Steps	Cost	Outcome
chi-discard-readfrom	go	go-chi/chi	#1110	2026-06-14	medium	bug fix	medium	3	5	256	54	$0.14	✓
chrono-offset-minute-clamp	rust	chronotope/chrono	#1798	2026-06-22	large	bug fix	medium	6	4	222	58	$0.11	✓
hono-etag-star-match	typescript	honojs/hono	#5084	2026-07-10	large	bug fix	medium	3	4	137	34	$0.07	✓
hono-trie-empty-wildcard	typescript	honojs/hono	#5102	2026-07-10	large	bug fix	medium	3	5	213	74	$0.23	✓
hono-url-param-prefix	typescript	honojs/hono	#5096	2026-07-12	large	bug fix	medium	4	4	130	38	$0.06	✓
jiff-strftime-lenient	rust	BurntSushi/jiff	#606	2026-07-19	xlarge	bug fix	medium	5	3	734	192	$0.88	✓
more-itertools-subfactorial	python	more-itertools/more-itertools	#1166	2026-06-10	medium	feature	medium	4	3	74	70	$0.16	✓
packaging-licenseref-plus	python	pypa/packaging	#1219	2026-06-11	medium	bug fix	medium	4	4	41	34	$0.06	✓
pflag-custom-isboolflag	go	spf13/pflag	#487	2026-07-02	medium	bug fix	medium	3	4	252	82	$0.33	✓
regex-leftmost-suffix-candidate	rust	rust-lang/regex	#1364	2026-07-15	xlarge	bug fix	veryhard	7	5	1257	104	$2.05	✗ wrong answer
semver-tilde-prerelease-bound	javascript	npm/node-semver	#878	2026-06-19	medium	bug fix	medium	5	4	47	54	$0.10	✓
sqlglot-udtf-chained-alias	python	tobymao/sqlglot	#7828	2026-07-07	xlarge	bug fix	hard	4	3	87	58	$0.24	✓
time-strftime-truncated-padding	rust	time-rs/time	#782	2026-06-12	large	bug fix	medium	5	4	243	58	$0.16	✓

Ten distinct upstream projects. Every task records its upstream merge date in metadata, so the clean/contaminated split can be recomputed against a future model’s cutoff rather than re-derived by hand.

Harness defects fixed before these numbers were taken

An earlier version of this comparison read 7/13 vs 12/13 — a large gap pointing the wrong way for contamination. It was five stalled HTTP requests. Four defects were found and fixed; each changed scoring behaviour, so results published before them (Reports No. 4 and No. 08) are not comparable to this one.

Defect	Effect on a score	Fix
The run's remaining budget was passed as the per-request socket timeout, and supplying a timeout also bypassed the retry wrapper	A single stalled request consumed an entire task and scored it 0. Six of 13 tasks died this way in the first run; one spent 21 s working and 1785 s blocked on one socket.	The budget now caps the request instead of becoming it; retry is live on every path, bounded so retries cannot overrun the budget.
The request ceiling was calibrated from a pooled latency distribution	300 s sat below a response that had legitimately completed in 333 s, aborting real work. Round-trip time grows with context, so the tail is per-task, not pooled — maxima were 30 s on small repos versus 333 s on large ones.	Ceiling raised to 600 s: ~1.8× above the slowest real response, ~1.6× below the shortest observed stall.
A run failing on an infrastructure error was recorded and dropped	The suite reported a tidy pass@1 over a silently smaller denominator, with a different task dropping each run. Two consecutive runs both printed "over 12 tasks" for different reasons.	Environmental failures are re-queued (bounded, and gated by the spend cap) and surfaced in the summary.
`large` and `xlarge` shared an 1800 s budget despite xlarge allowing 33% more steps	Neither tier could spend its step allowance at the 10–15 s/step measured on large repos, so runs were cut off mid-work — penalising whichever arm carried more big repos.	Raised to 2700 s / 3600 s so the budget covers the allowance.

Caveats

Near ceiling. At 11–12 of 13 the model solves nearly everything it attempts, and a ceiling leaves memorisation little room to show itself. This rules out a large contamination effect, not a small one. A sharper test needs harder tasks or a weaker model, not more tasks at this difficulty.

Sample size. 13 tasks per arm; one task is 7.7 pp. Only multi-task gaps are meaningful.

Single attempt. One run per task, no pass@k. Borderline tasks may flip between runs.

One model. Contamination sensitivity is not necessarily uniform across model families.

Emulation. These runs executed amd64 sandbox images on an arm64 host (~2× overhead on tool calls). Scores are unaffected; wall-clock and cost-per-solved are inflated relative to native runs and are not comparable to earlier reports.

Not blinded. Task selection for the post-cutoff arm was made by the same process that ran the comparison.

Reproduction

Both arms are subsets recorded in the committed suite manifests — `pre-2026-05` in `tasks/v3/suite.json` and `new_in_v4` in `tasks/v4/suite.json`. Materialise each as a scratch suite of symlinks, then:

```
vulcanbench run --suite v3-pre --model anthropic:claude-opus-5 --effort medium --no-judges --repeat 1 --max-concurrency 4 --max-cost 30.00
vulcanbench run --suite v4-new --model anthropic:claude-opus-5 --effort medium --no-judges --repeat 1 --max-concurrency 4 --max-cost 30.00
```

Requires the harness fixes above. Pricing: Claude Opus 5 $5/$25 per million tokens. Full traces, final patches and replay HTML for every run are under `runs/`. Machine-readable results: `docs/results/v4-contamination-2026-07/results.json`.