Does training-data contamination move Claude Opus 5's score?

A controlled A/B: thirteen v3 tasks whose upstream fixes merged before Opus 5's May 2026 training cutoff (the model may have memorised them) against thirteen new v4 tasks merged after it (it cannot have) — run back-to-back with identical settings.

Abstract

No detectable contamination effect. The arm Opus 5 could have memorised scores 11/13; the arm it cannot have seen scores 12/13 — and each side records exactly one genuine wrong answer. The visible one-task gap is repo-scale composition, not memory: the contaminated arm carries 11 of 13 large-or-xlarge repositories against the clean arm's 8, which is also why it cost 3.6× more and ran 2× longer per task. At 11–12 of 13 the model is near ceiling, so this rules out a large effect, not a small one. Getting to this null took four harness fixes: before them, the same comparison read 7/13 vs 12/13 — a large gap pointing the wrong way for contamination, produced entirely by stalled HTTP requests scored as failures.

Table 1. Results (Claude Opus 5, medium effort, one attempt per task)

Arm	Score	pass@1	Genuine misses	Hit wall-clock	Cost	Tok/task	Time/task	$/task	$/solved
Clean (post-cutoff)	**12/13**	**92.3%**	**1**	**0**	**$4.58**	**38 K**	**4.7 min**	**$0.35**	**$0.38**
Contaminated (pre-cutoff)	11/13	84.6%	1	1	$16.36	121 K	9.4 min	$1.26	$1.49

Genuine misses completed and produced a wrong patch. Hit wall-clock = still working productively when the budget expired — a different failure mode, not a statement about capability. Zero stalls, zero infrastructure errors, zero retries on either arm.

Findings

1. No detectable contamination effect. Genuine capability misses are 1 versus 1. The headline gap is a single task (7.7 pp), inside the noise floor at n=13. Memorisation of the upstream fix did not measurably help.

2. The visible gap is repo-scale composition. The contaminated arm carries 11/13 large-or-xlarge repositories against the clean arm's 8/13. Bigger repos cost more per step — context-heavy round trips, slower test runs — which is why the same 13-task shape cost 3.6× more and ran 2× longer per task, and why the one wall-clock loss landed there. Any cross-suite comparison must report scale mix alongside score.

3. Evidence against a large effect, not proof of none. At 11–12 of 13 the model is near ceiling, and a ceiling leaves memorisation little room to show itself. A sharper test needs harder tasks or a weaker model, not more tasks at this difficulty.

4. Most of the apparent signal was harness bugs. Before the fixes below, the comparison read 7/13 vs 12/13. Fixing the request ceiling and retries moved it to 10/13 vs 12/13; all four fixes landed it at 11/13 vs 12/13. Each fix moved the contaminated arm up and the arms closer together — the "contamination signal" was infrastructure.

Table 2. Harness defects found and fixed for this report

Defect	Effect on scores
Run budget passed as the per-request socket timeout, disabling retries	One stalled request consumed a whole task and scored it 0; six of 13 first-run tasks died this way
Request ceiling set from a pooled latency distribution	300 s sat below a response that legitimately completed in 333 s; now 600 s
Infrastructure errors dropped the task instead of retrying	Tidy pass@1 over a silently smaller denominator, a different task each run
large and xlarge shared an 1800 s budget	Neither tier could spend its step allowance; now 2700 s / 3600 s

These changed scoring behaviour. Results published before them — including Reports No. 04 and No. 08 — were produced under the old semantics and are not comparable to this report.

Table 3. Failure map (the three tasks not solved; the other 23 passed)

Task	Arm	Scale	Outcome
sqlglot-canonicalize-internal-names	contaminated	large	× completed, wrong patch (1744 s, 298 steps)
networkx-leiden-communities	contaminated	xlarge	× still working at the 3600 s budget (274 steps)
regex-leftmost-suffix-candidate	clean	xlarge	× completed, wrong patch (1257 s, 104 steps)

Method & caveats

Both arms graded by deterministic hidden tests in a network-isolated Docker sandbox; every task pre-validated gold = 1.0, unpatched = 0.0. The arms are committed suite manifests — pre-2026-05 in tasks/v3/suite.json, new_in_v4 in tasks/v4/suite.json — and every task records upstream_merged, so the split can be recomputed against any future model's cutoff. v4 is clean for a May 2026 cutoff specifically, not indefinitely. Caveats: n=13 per arm, one task is 7.7 pp, so only multi-task gaps are meaningful; single attempt per task; one model — contamination sensitivity is not necessarily uniform across families; wall-clock inflated ~2× by amd64 emulation on an arm64 host (scores unaffected). Opus 5 at $5/$25 per million tokens.