VulcanBench · Technical Report No. 6

Does it work every time? Reliability under repeated runs

Claude Opus 4.8 (low effort) vs Claude Haiku 4.5, five runs per task on a suite with a frontier-hard tier

2026-07-07 · 15 evals · 135 scored runs · $79.55 total · deterministic hidden-test grading

Abstract. Single-run benchmarks measure luck-adjusted capability; repeated runs measure dependability, and the two rank models differently. On pass@1 the frontier model and the budget model look close (0.600 vs 0.667). On **pass^5 — solved in all five runs — the cheapest model wins: 0.667 vs 0.533**. Haiku 4.5 is perfectly bimodal across 64 runs: ten tasks solved every single time, including the multi-site Flask teardown redesign that has stopped Fable 5, Opus 4.8, and Sonnet 5 in every prior run, and five frontier-hard tasks never solved. Opus 4.8 is the flaky one: two of its passing tasks are coin flips that every previous single-run report scored as clean 1.0s. The frontier-hard tier holds for both models: zero solves in thirty attempts.

1. Results

Model	pass@k (≥1 win)	pass@1	pass^k (all wins)	Flaky tasks	Runs	Cost
Claude Opus 4.8 (low)	0.667	0.600	0.533	2	71	$46.98
Claude Haiku 4.5	0.667	**0.667**	**0.667**	**0**	64	**$32.49**

Per-task repeats: Opus n=5 (two shortfalls noted in §4); Haiku n=5 on anchors, n=3 on the hard tier by design. pass@1 is the mean per-task solve rate; pass^k requires solving a task in every attempt.

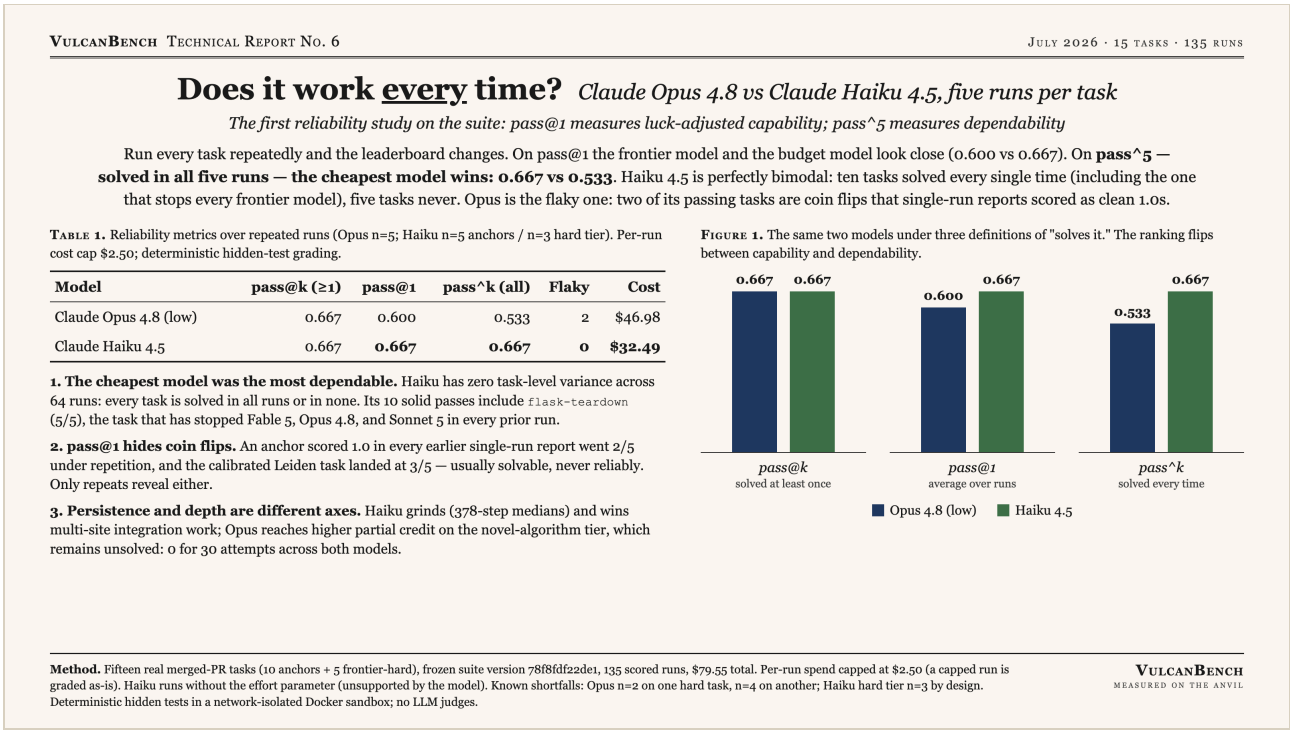

Figure 1. The same two models under three definitions of "solves it." The ranking flips between capability and dependability.

2. Findings

1. **The cheapest model was the most dependable.** Haiku shows zero task-level variance across 64 runs: every task is solved in all attempts or in none, so its pass@1, pass@k, and pass^k coincide at 0.667. Opus solves fewer tasks dependably (8) than Haiku (10), because two of its passes are unreliable.

2. **pass@1 hides coin flips.** `sqlglot-iso8601-nanos` scored 1.0 in every earlier single-run report; under repetition it is 2/5 for Opus-low. The calibrated Leiden task landed at 3/5 — a genuine middle-band task, usually solvable, never reliably. Neither behavior is visible at n=1.

3. **Persistence and depth are different axes.** Haiku grinds (378-step medians) and that persistence reliably closes multi-site integration work — `flask-teardown` 5/5 where every frontier model has failed, `aiohttp-upgrade-deferred` 5/5 where Fable refuses. But it hits the novel-algorithm tier as hard as everyone else (0/15) with lower partial credit than Opus (canonicalize flat 0.17 vs Opus reaching 0.50; Leiden 0/3 vs 3/5).

4. **The frontier-hard tier holds.** Zero solves in 30 attempts across both models. Every configuration ever run against the tier — Fable 5, Opus 4.8, Sonnet 5, and now Haiku 4.5 — has failed it.

3. Per-task detail

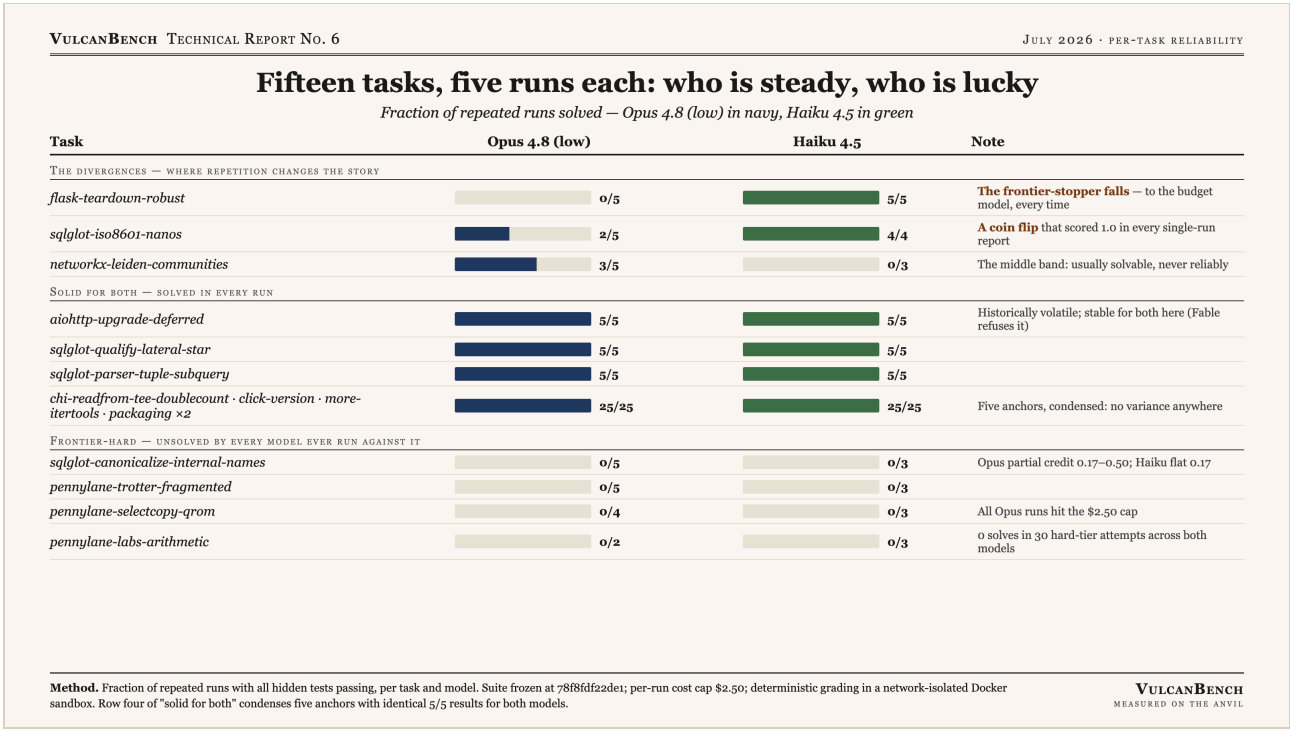

Figure 2. Fraction of repeated runs solved, per task and model, grouped by behavior.

4. Method, caveats, and cost

Fifteen tasks from real merged pull requests (10 anchors + 5 frontier-hard), all merged after model training cutoffs, graded by deterministic hidden tests in a network-isolated Docker sandbox; no LLM judges. Suite frozen at version `78f8fdf22de1`; every run records the task hash it was scored against. Per-run agent spend capped at $2.50 (a capped run is graded as-is and reported as a bounded DNF); per-suite budget breakers; Haiku runs without the effort parameter, which the model rejects.

Coverage shortfalls, stated plainly: Opus has n=2 on `pennylane-labs-arithmetic` and n=4 on `pennylane-selectcopy-qrom` (all four qrom runs hit the $2.50 cap); Haiku has n=4 on `iso8601-nanos` and n=3 on the hard tier by design. Budget breakers stop launching new runs but let in-flight runs finish; the Opus pass overshot its $38 cap to $43.13. Total spend for the study was $79.55 across 135 scored runs, made affordable by per-run caps, budget breakers, and gap-filling resume (`--only-missing`).

VulcanBench — open-source coding benchmark. Measured on the anvil, graded by hidden tests.
github.com/morganlinton/VulcanBench