VulcanBench · Technical Report No. 5

Fable 5 vs Opus 4.8 on a frontier-hard tier

Two models, two effort levels, on a suite built to un-saturate the frontier

2026-07-04 · 15 evals · 60 runs · 3 refusals · deterministic hidden-test grading · corrected costs

Abstract. Five net-new-algorithm tasks were added to the VulcanBench suite specifically to defeat low-effort frontier models. On the thirteen tasks every configuration completed, binary pass@1 is a **four-way tie: 7 of 13 in every cell** — all four configurations pass the same seven tasks and fail the same six. The signal has moved off the binary metric. On the five frontier-hard tasks, partial credit **rises with reasoning effort for Opus 4.8 (0.26 → 0.42)** and **collapses for Fable 5 (0.17 → 0.00)**: at high effort Fable spends its entire output budget deliberating and writes no code. Opus delivers the tied score at roughly **one-third of Fable's cost**, and Fable alone refused two legitimate networking tasks as safety false-positives.

1. Results

Model (effort)	Pass@1	Hard tier	Cost	Time	Steps	Refusals
Claude Fable 5 (low)	7/13	0.17	$22.35	104 min	982	2
Claude Fable 5 (high)	7/13	0.00	$26.27	99 min	1186	1
Claude Opus 4.8 (low)	7/13	0.26	**$6.95**	**37 min**	1316	0
Claude Opus 4.8 (high)	7/13	**0.42**	$12.77	54 min	1698	0

Aggregate on the 13-task common set (8 anchors + 5 hard-tier), one run per task, deterministic grading. *Hard tier* is the mean fraction of hidden tests passed across the five frontier-hard tasks. The two tasks Fable refused are excluded from all four configurations so the comparison is like-for-like. Costs reflect provider prompt-cache discounts.

VulcanBench Technical Report No. 5 — July 2026 · 15 tasks · 60 runs

Claude Fable 5 *(low / high)* vs. Claude Opus 4.8 *(low / high)*

Can reasoning effort crack a frontier-hard tier? Real merged-PR tasks, deterministic hidden-test grading

On the 13 tasks every configuration completed, binary pass@1 is a **four-way tie: 7/13 in every cell**, with all four configurations passing the same seven tasks and failing the same six. The signal lives elsewhere: on the five frontier-hard tasks, partial credit **rises with effort for Opus (0.26 → 0.42)** and **collapses for Fable (0.17 → 0.00)** — at high effort Fable exhausts its output budget thinking and writes no code at all. Opus delivers the tied score at roughly **one-third of Fable's cost**, and Fable alone refused two legitimate networking tasks outright.

Table 1. Apples-to-apples aggregate on the 13-task common set (8 anchors + 5 hard-tier), one run per task, deterministic grading. Two tasks Fable refused are excluded from all configurations.

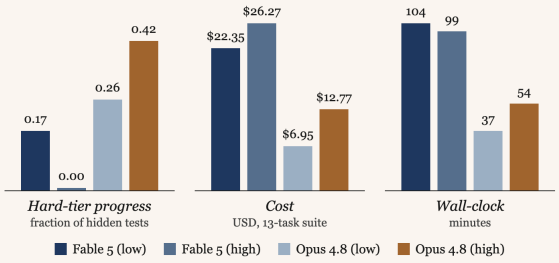
Model (effort)	Pass@1	Hard tier	Cost	Time	Refusals
Claude Fable 5 (low)	7/13	0.17	$22.35	104 min	2
Claude Fable 5 (high)	7/13	0.00	$26.27	99 min	1
Claude Opus 4.8 (low)	7/13	0.26	**$6.95**	**37 min**	**0**
Claude Opus 4.8 (high)	7/13	**0.42**	$12.77	54 min	**0**

1. Effort is a one-way lever. More thinking lifts Opus up the hard tier (its best task climbs 1/6 → 5/6 of hidden tests) and drops Fable to zero: at high effort Fable burned its full output budget deliberating and never wrote the implementation.

2. Same score, a third of the price. Every configuration solves exactly the same seven tasks; Opus does it for a third of Fable's cost in half the wall-clock.

3. Guardrails cost real points. Fable refused two legitimate networking tasks as cyber-risk false positives; Opus completed both. Unfiltered, this masquerades as a capability gap.

Figure 1. The three axes that separate a four-way pass@1 tie. Hard tier = mean fraction of hidden tests passed on the five frontier-hard tasks.



Method. Fifteen tasks from real merged PRs (flask, sqlglot, click, more-itertools, packaging, networkx, pennylane; all merged after model training cutoffs), including five frontier-hard tasks individually screened to defeat low-effort frontier models. Deterministic hidden tests in a network-isolated Docker sandbox; one run per task per configuration; no LLM judges. Two tasks Fable refused (cyber-classifier false positives) are excluded from all four configurations for the apples-to-apples table. Costs reflect provider prompt-cache discounts. Total spend ≈ $100.

VulcanBench — Open-source coding benchmark

Figure 1. The three axes — hard-tier progress, cost, wall-clock — that separate a four-way pass@1 tie.

2. Findings

1. **Frontier pass rates have converged to a literal tie.** All four configurations score 7/13, passing the same seven tasks and failing the same six. An eval reporting only binary pass@1 would call these configurations identical — which is precisely why a harder tier and a partial-credit metric are needed to recover any signal.

2. **Reasoning effort is a one-way lever.** More thinking compounds for Opus (hard-tier score 0.26 → 0.42; its best task climbs from 1/6 to 5/6 of hidden tests) and backfires for Fable (0.17 → 0.00). At high effort Fable exhausted its output budget deliberating and produced no implementation on every hard task. "More effort" is a model-specific behavior, not a universal upgrade.

3. **At equal accuracy, the efficiency gap is about 3x.** Opus matches Fable's score at a third of the cost ($6.95–$12.77 versus $22.35–$26.27) and about half the wall-clock. When correctness ties, cost and latency are the benchmark.

4. **Safety guardrails masquerade as capability loss.** Fable refused two legitimate tasks — an HTTP-upgrade handler and an `io.Tee` bug — as cyber-risk false positives; Opus refused none. On raw scores that reads as "Fable failed two tasks." Refusal-adjusted scoring should be table stakes for model comparisons on systems and networking code.

5. **Models follow training priors over explicit instructions.** On the Leiden task the objective function is spelled out in the prompt, formula included, with a literal "this is CPM, not modularity" warning. The model implemented modularity anyway, scoring far below threshold. Tasks graded against a specified-but-unusual convention expose an instruction-versus-prior failure mode that standard benchmarks never touch.

3. The hard tier, task by task

Binary pass@1 records all five frontier-hard tasks as failures for every configuration. Partial credit — the fraction of each task's hidden tests passed — is where the four configurations come apart.

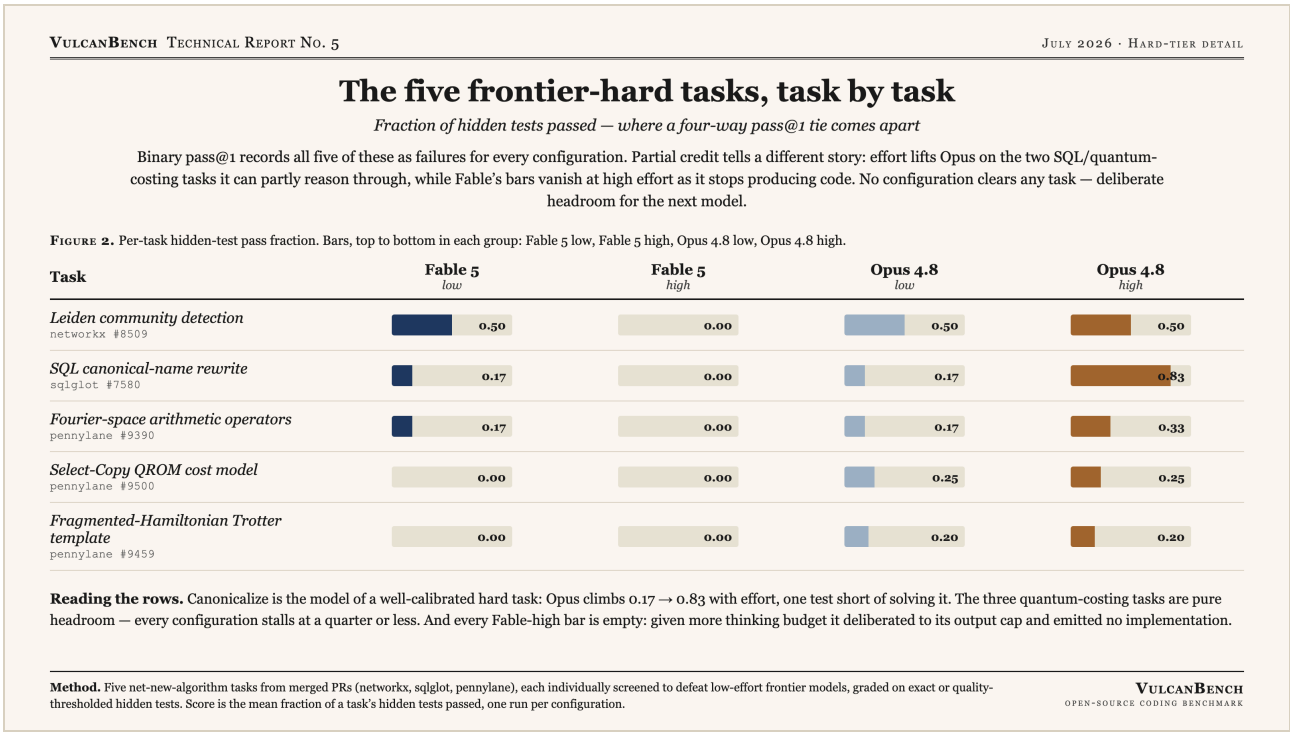


Figure 2. Per-task hidden-test pass fraction across the four configurations.

SQL canonical-name rewrite is the model of a well-calibrated hard task: Opus climbs from 0.17 at low effort to 0.83 at high, one test short of solving it, while Fable sits at 0.17 then 0.00. **Leiden community detection** holds steady at 0.50 for three of four cells — a partial implementation that satisfies structure and determinism but misses the quality bar. The three **quantum resource-estimation** tasks (Fourier-space arithmetic, Select-Copy QROM, and the fragmented-Hamiltonian Trotter template) are pure headroom: every configuration stalls at a quarter of the hidden tests or less. And every Fable-high bar is empty — given more thinking budget, it deliberated to its output cap and emitted no implementation.

4. Method & calibration

Fifteen tasks from real merged pull requests (flask, sqlglot, click, more-itertools, packaging, networkx, pennylane; Python and Go), all merged after model training cutoffs, graded by deterministic hidden tests in a network-isolated Docker sandbox. One run per task per configuration; no LLM judges. Five of the fifteen are net-new-algorithm tasks individually screened to defeat low-effort frontier models: a native Leiden implementation (networkx #8509),

a SQL canonical-name rewrite (sqlglot #7580), and three quantum resource-estimation templates (pennylane #9390, #9500, #9459).

The hard tier was calibrated after an initial run showed it graded partly on conventions a correct implementation could not derive from the task description. Grading was made fair-hard rather than merely difficult: the Leiden task moved from exact-partition matching to semantic quality thresholds (the constant-Potts-model score on two graphs, set below the reference implementation's own across-seed floor and far above degenerate baselines) plus the algorithm's defining connectivity guarantee; the three quantum tasks kept exact-output grading but gained worked acceptance examples with exact reference values at parameter points distinct from the hidden graded cases. Difficulty was preserved; unknowability was removed. Post-calibration, the remaining failures are attributable to genuine capability gaps — most sharply, a model writing modularity-style communities after being explicitly told the objective was not modularity.

Two tasks Fable refused as cyber-classifier false positives (an aiohttp HTTP-upgrade handler and a chi `io.Tee` double-count bug) are excluded from all four configurations in the apples-to-apples table so the comparison is like-for-like; the refusals are reported separately as a reliability finding. Costs reflect each provider's automatic prompt-cache discount. Total spend for the run was approximately $100.

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