Reasoning-effort scaling on the cost-quality frontier: an evaluation of Claude Sonnet 5 and Opus 4.8 on VulcanBench v1

VulcanBench Project
Open-source LLM coding benchmark

July 1, 2026

Abstract. Functional pass/fail on software-engineering benchmarks has saturated at the frontier: leading models solve nearly identical task sets, so raw accuracy no longer separates them. We argue that reasoning effort, the model's thinking-token budget, is an inexpensive lever that re-introduces measurable separation without authoring harder tasks. Holding a fixed 52-task suite constant, we sweep two Anthropic models, Claude Sonnet 5 and Claude Opus 4.8, across three normalized effort levels at three repeats each (936 runs), grading every run with deterministic hidden tests. We report accuracy (pass@1), cost per solved task, token usage, and latency, and read the outcome as a cost-quality Pareto frontier. Sonnet 5 attains a perfect pass@1 of 100% at high effort, the only configuration to solve all 156 of its runs, and it dominates the frontier: at matched accuracy it is 39% cheaper per solved task than Opus 4.8, and at high effort it is at once more accurate (by 2.6 points) and 7.7% cheaper. The two models scale with effort very differently. Sonnet 5 is effort-elastic, converting a 96% increase in thinking tokens into a monotonic accuracy gain concentrated on hard tasks, whereas Opus 4.8 is effort-inelastic, its accuracy flat within noise. We discuss the implications for model selection and for using effort as a discriminating axis on near-saturated benchmarks.

Keywords: code generation, LLM evaluation, reasoning effort, cost efficiency, Pareto frontier, benchmark saturation.

1 Introduction

Contemporary code-generation benchmarks increasingly fail to separate frontier models on their headline metric. On functionally graded suites, the strongest models pass and fail almost the same tasks, and the residual differences fall inside run-to-run noise. Cognition report the same effect from the opposite direction, stating that "correctness is now table stakes" and moving their difficulty into maintainer-style mergeability rubrics [1]. The usual response, authoring ever-harder tasks, is costly, and well-known hard problems are quickly memorized by the next model generation.

We take a different lever. Reasoning effort, the budget a model is permitted to spend thinking before it answers, is a first-class, normalized control on current Anthropic models. Our hypothesis is simple: if a suite is saturated at high effort, then *starving* the thinking budget re-exposes the differences the ceiling hides, and the accompanying cost, token, and latency measurements are always informative because they differ even when accuracy does not. In effect, an effort sweep un-saturates a fixed suite for free, with no new task authoring.

We instantiate this idea on VulcanBench v1, a 52-task suite graded by deterministic hidden tests, and sweep Claude Sonnet 5 and Claude Opus 4.8 across low, medium, and high effort at three repeats per task, for 936 runs in total. Our contributions are:

- an effort-sweep evaluation protocol that turns a saturated, deterministically graded suite into a cost-quality Pareto measurement;
- a 936-run comparison of Sonnet 5 and Opus 4.8 showing that Sonnet 5 occupies the entire frontier and reaches a perfect 100% pass@1 at high effort;
- evidence that the two models differ sharply in *effort elasticity*, and a localization of Sonnet 5's gains to the hard-difficulty tier; and
- practical model-selection guidance derived from cost per solved task rather than raw accuracy.

2 Related work

SWE-bench established deterministic, test-based grading of real GitHub issues and remains the template for reproducible code evaluation [3]. FrontierCode [1] argues that functional correctness has saturated and scales difficulty with quality rubrics rather than patch size, grading whether a maintainer would merge a change. CursorBench [2] grades real agent sessions with adaptive, agentic checks and terse prompts. Model cards [4] set the reporting discipline we follow for a single, scoped comparison. Our study is complementary: we retain deterministic grading, as in SWE-bench, but move the discriminating axis away from task difficulty and toward reasoning effort and cost, which remain informative even where correctness is saturated.

3 Experimental setup

3.1 Suite and grading

VulcanBench v1 comprises 52 self-contained software-engineering tasks spanning Python, Go, TypeScript, and Rust, with a difficulty mix of 12 easy, 21 medium, and 19 hard tasks. Each task ships a hidden test suite; a run is scored functional, and counts as solved, if and only if those tests pass (functional = 1.0). No LLM judge or rubric is involved, so the accuracy axis is fully deterministic and reproducible.

3.2 Models and pricing

We evaluate `claude-sonnet-5` and `claude-opus-4-8`. Reported costs use standard list prices of 3/15 USD per million input/output tokens for Sonnet 5 and 5/25 for Opus 4.8 [5]. The two models share a tokenizer, so token counts are directly comparable between them.

3.3 Effort levels

Each model is run at three normalized effort levels (low, medium, high). Effort names are not equal-thinking across model families, so we report token counts alongside effort labels; the per-task step budget is held identical across effort cells so that only the thinking budget varies.

3.4 Design and metrics

The full design is 2 models x 3 efforts x 52 tasks x 3 repeats = 936 runs, executed in a Docker sandbox. We report pass@1, the mean over tasks of the fraction of a task's attempts that pass, with standard error across tasks; cost per solved task, total spend divided by solved runs; and mean tokens, latency, and agent steps per run. The complete sweep consumed 22.2M tokens and 105.26 USD at list price.

Table 1. Per-configuration results on VulcanBench v1 (n = 156 runs per cell; pass@1 with standard error across 52 tasks). Best value in each column in bold. Costs at standard list price.

Model	Effort	pass@1	s.e.	$/solved	$ total	Tokens	Lat. (s)	Steps
Opus 4.8	low	97.4	2.02	0.115	17.42	18,406	41.3	31.0
	medium	98.1	1.42	0.127	19.40	20,461	42.9	32.5
	high	97.4	2.02	0.142	21.61	22,726	46.3	33.9
Sonnet 5	low	97.4	2.02	**0.070**	10.57	18,130	45.0	29.2
	medium	99.4	0.64	0.102	15.78	27,437	50.6	36.4
	high	**100.0**	**0.00**	0.131	20.47	35,460	55.7	42.0

4 Results

Table 1 and Figure 1 summarize the sweep. Three results stand out. First, Sonnet 5 *owns the entire cost-quality Pareto frontier*: its three configurations are the only non-dominated cells, and all three Opus 4.8 configurations lie below and to the right of the frontier. Second, Sonnet 5 at high effort solves every one of its 156 runs (pass@1 = 100.0%, standard error 0.00), the only cell in the matrix to do so; Opus 4.8 peaks at 98.1% and never solves every run. Third, at matched accuracy Sonnet 5 is dramatically cheaper: at 97.4% pass@1, Sonnet 5 (low effort) costs 0.070 USD per solved task against Opus 4.8 (low effort) at 0.115, a 39% saving. At the top of the frontier, Sonnet 5 (high) is simultaneously more accurate than Opus 4.8 (high), by 2.6 points, and 7.7% cheaper per solved task, a clean two-axis domination.

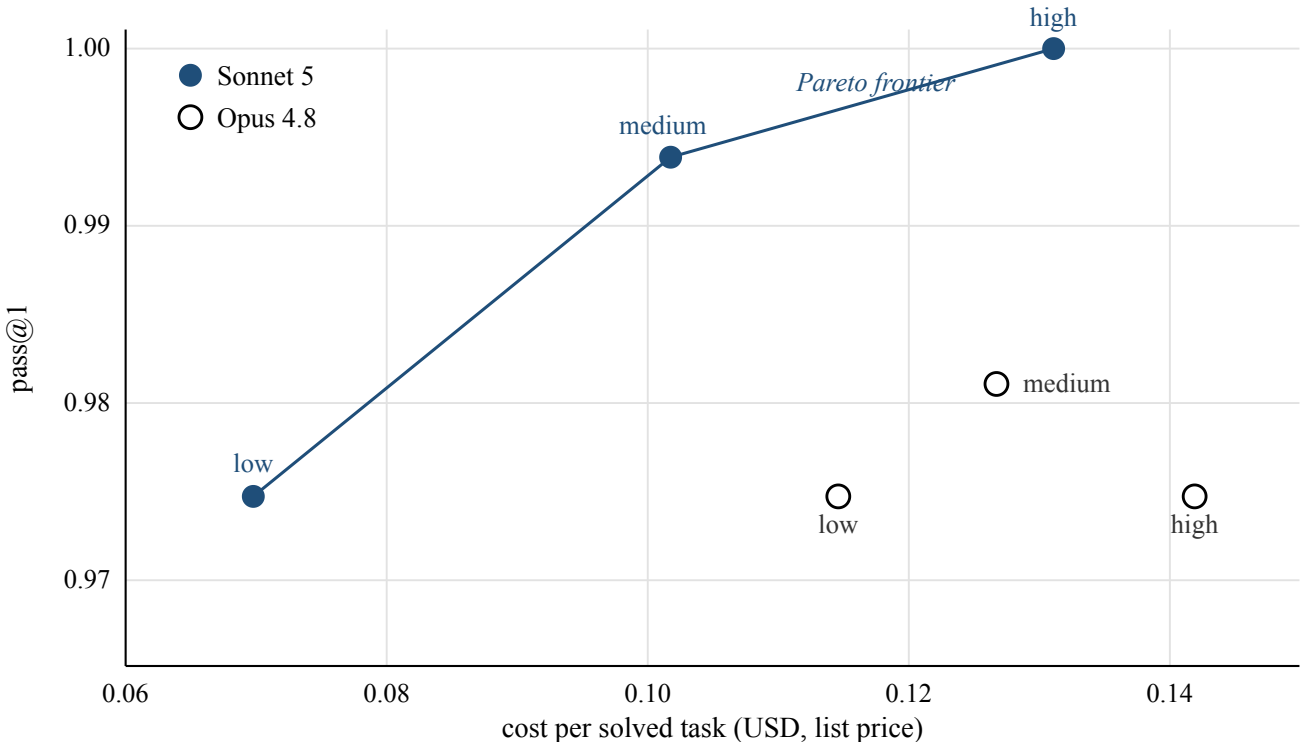

Figure 1. Cost-quality frontier. Filled markers denote Sonnet 5, open markers Opus 4.8; each point is labeled by its effort level. The three Sonnet 5 points (connected) form the Pareto frontier; every Opus 4.8 point is dominated. Up and to the left is better. The vertical axis is truncated to 0.965-1.001 to resolve sub-3-point differences.

5 Analysis

5.1 Effort elasticity

The models respond to effort in opposite ways (Figure 2). From low to high effort, Sonnet 5 gains 2.6 points of pass@1 (97.4 to 100.0) monotonically, while its mean thinking tokens rise 96% (18.1k to 35.5k) and its steps grow from 29 to 42. Opus 4.8, by contrast, is net-flat: pass@1 moves 97.4 to 98.1 to 97.4, inside its standard error, while tokens rise only 23%. On this suite the extra Opus budget buys tokens, not solutions. We term Sonnet 5 effort-elastic and Opus 4.8 effort-inelastic.

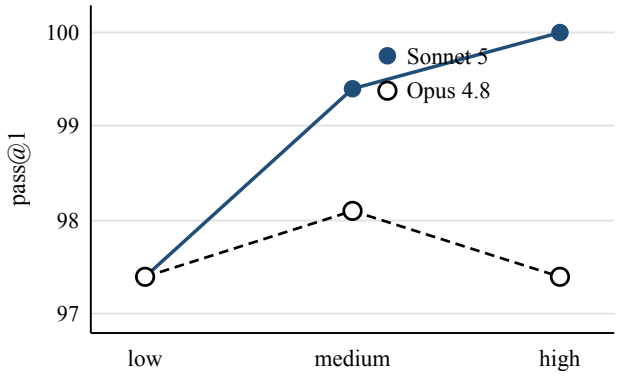

Figure 2. Effort elasticity. Sonnet 5 (solid) climbs monotonically to 100% as effort rises; Opus 4.8 (dashed) is flat within noise.

5.2 Where effort helps

Table 2 localizes the effect by task difficulty. Easy tasks are saturated for both models at every effort (36/36 runs). All of the movement is on medium- and hard-difficulty tasks. Sonnet 5's hard-tier pass climbs 54 to 56 to 57 of 57 runs as effort rises, and its effort literally converts specific failures into passes: `rs-borrow-split` is solved from medium effort upward, and `py-bytecode-vm` is solved at high effort. Opus 4.8 is flat on both tiers. Notably, at low effort Opus is marginally ahead on the hard tier (56 vs 54 of 57), but because it does not improve with effort, Sonnet 5 overtakes it by high effort (57 vs 56).

Table 2. pass@1 by task difficulty, as runs solved / runs attempted (3 repeats per task). Task counts in parentheses.

	Opus 4.8			*Sonnet 5*		
Difficulty	**low**	**med**	**high**	**low**	**med**	**high**
easy (12)	36/36	36/36	36/36	36/36	36/36	36/36
medium (21)	60/63	60/63	60/63	62/63	63/63	63/63
hard (19)	56/57	57/57	56/57	54/57	56/57	57/57

5.3 Cost efficiency

Because GLM-style token bloat and retries can erase a low per-token price, we treat cost per solved task as the honest unit. On this axis Sonnet 5 is cheaper than Opus 4.8 in every head-to-head comparison at equal or higher accuracy. The cheapest solved task overall is Sonnet 5 at low effort (0.070 USD); the cheapest route to a perfect run is Sonnet 5 at high effort (0.131

USD), still below Opus 4.8 at high effort (0.142) despite Sonnet's larger token footprint, because its per-token price is lower and its solved fraction is higher.

5.4 Latency

The one axis on which Sonnet 5 trails is wall-clock latency: it runs roughly 3 to 9 seconds slower per task on average, the cost of its larger reasoning traces. For batch evaluation this is immaterial; for tight interactive loops it is the trade the buyer makes for either lower spend (low effort) or higher accuracy (high effort).

6 Discussion

Two implications follow. For benchmark design, effort sweeping is a cheap way to recover discriminating power from a saturated suite: the low-effort dropoffs below the high-effort ceiling are real signal, and cost, tokens, and latency separate models even when accuracy does not. For model selection on this workload, the guidance is concrete. Run Opus 4.8 at low effort, since higher effort adds cost without accuracy. Dial Sonnet 5 effort up to the accuracy target: low effort matches Opus at a 39% lower cost per solved task, and high effort reaches 100% while remaining cheaper than Opus at high effort. Across the entire frontier, Sonnet 5 is the better choice.

7 Limitations

Our grading is correctness-only and does not measure code quality or mergeability, the axis FrontierCode targets [1]; a model could pass tests with code a maintainer would reject. The suite is near-saturated (both models exceed 97%), so the accuracy separations are small in absolute terms and the cost, effort, and latency axes carry most of the signal. Part of Opus 4.8's residual error is the task `rs-borrow-split`, a known spec-ambiguous item, so its true capability gap is slightly smaller than Table 1 suggests. Effort names are not equal-thinking across model families; we mitigate this by reporting tokens alongside effort. Both models are from one vendor, which aids token comparability through a shared tokenizer but limits external validity against other providers. Finally, we use three repeats and list prices; Sonnet 5's introductory promotion (2/10 USD per million tokens through 2026-08-31) lowers its current cost by roughly a third and would widen its lead, so our figures are conservative.

8 Conclusion

Sweeping reasoning effort re-introduces separation on a suite that functional pass/fail can no longer discriminate. On VulcanBench v1, Sonnet 5 owns the cost-quality Pareto frontier, reaches a perfect 100% at high effort, and is uniquely effort-elastic, while Opus 4.8 is effort-inelastic. For this workload, and at every accuracy target it supports, Sonnet 5 is the more cost-effective model.

References

1. Cognition. FrontierCode: measuring frontier coding ability with mergeability rubrics. cognition.com/blog/frontier-code, 2026.
2. Cursor. CursorBench: evaluating coding agents on real sessions. cursor.com/blog/cursorbench, 2026.
3. C. Jimenez, J. Yang, et al. SWE-bench: Can language models resolve real-world GitHub issues? ICLR, 2024. arXiv:2310.06770.
4. M. Mitchell, S. Wu, et al. Model cards for model reporting. Proc. FAT*, 2019.
5. Anthropic. Claude model overview and pricing. anthropic.com, 2026.

Appendix A Reproduction

The harness does not read `.env`; export keys before running. Grading uses the Docker sandbox to avoid local pytest configuration leaking into task verification.

```
export ANTHROPIC_API_KEY="..."
vulcanbench effort-sweep --suite v1 \
  -m anthropic:claude-sonnet-5 \
  --efforts low,medium,high --repeat 3 \
  --sandbox docker --no-judges -o ../vb-effort
# repeat with -m anthropic:claude-opus-4-8

python scripts/effort_pareto.py ../vb-effort
vulcanbench report -o ../vb-effort
```

All numbers in this report derive from the 936 run summaries under `../vb-effort` (52 tasks x 2 models x 3 efforts x 3 repeats).