Grok 4.6 in Grok Build vs. Cursor vs. a bare-bones harness

The second model-times-harness study on VulcanBench, and the first three-way one: one model, three delivery systems, on twenty-three real merged post-cutoff PRs. Grok Build is xAI's own agent CLI; Cursor is Report No. 15; the bare-bones reference loop is Report No. 14.

Abstract

Inside xAI's own harness the effort knob finally does what it promises. Grok 4.6 in Grok Build climbs 84.1, 88.4, 89.9, 92.8 across low, medium, high and xhigh: monotone, every step up buying accuracy. Neither other system behaves that way. Through the raw API the same knob is non-monotone and collapses at high (87.0 to 73.9); inside Cursor it is nearly flat (84.1 to 88.4 across a 4.3-point spread). The 92.8% at xhigh is the highest pass@1 VulcanBench has recorded for Grok 4.6 on this suite in any harness, 14.5 points above the raw API at the same setting and 7.3 above Cursor's best. Grok Build is also the fastest of the three at every level, finishing a high-effort task in a median 4.5 minutes against the API's 16.3. The gains are not free: token use nearly triples from low to xhigh, and the run cost at list-price API rates rises from $28.93 to $75.13 per level.

Table 1. The three systems

	A: bare-bones	B: inside Cursor	C: Grok Build
Path	xAI API, minimal loop	cursor-agent CLI	grok CLI 1.0.5, grok.com plan
Effort control	reasoning_effort per request	baked into the model id	--reasoning-effort, read back from the session receipt
Observability	tokens and dollars	none reported	full token split, per-run cost
Attempts	1 per task (Report No. 14)	3 per task (Report No. 15)	3 per task, 277 runs, 0 contaminated

Table 2. pass@1 by nominal effort level (mean per-task success rate; ±1 stderr)

Effort	Grok Build	In Cursor	Bare-bones	GB time/task	Cursor	Bare-bones
low	84.1% ±6.9	84.1% ±7.5	82.6% ±8.1	2.4 min	3.5 min	4.8 min
medium	88.4% ±5.4	84.1% ±7.2	87.0% ±7.2	4.3 min	4.6 min	14.2 min
high	89.9% ±5.3	88.4% ±6.5	73.9% ±9.4	4.5 min	6.9 min	16.3 min
xhigh	**92.8% ±5.1**	**85.5% ±7.2**	**78.3% ±8.8**	**4.1 min**	**6.5 min**	**15.8 min**

Grok Build columns: 69 to 70 runs each (three attempts per task, 23/23 tasks, zero contaminated on either audit channel). Cursor: 69 runs each. Bare-bones: 23 runs each, single attempt. Times are medians and include provider latency. Grok Build clears the bare API's error bars at high and xhigh; its margin over Cursor sits inside them at every level.

Table 3. Grok Build cost and token use per effort level (23 tasks, three attempts)

Effort	pass@1	pass@3	Prompt tokens	Completion	API-equivalent
low	84.1%	91.3%	40.4M	454K	$28.93
medium	88.4%	95.7%	80.6M	986K	$55.71
high	89.9%	95.7%	92.7M	1.20M	$63.88
xhigh	**92.8%**	**95.7%**	**112.4M**	**1.38M**	**$75.13**

Cost is counterfactual value at xAI list prices, not cash: these runs billed a grok.com subscription, so marginal cash was zero. Prompt tokens include cache reads, which dominate agentic loops. The whole sweep is $223.65 of API-equivalent value.

Findings

1. The effort knob works, and only here. 84.1, 88.4, 89.9, 92.8 is monotone across all four settings, an 8.7-point climb. The raw API over the same knob is non-monotone with a collapse at high; Cursor is flat within 4.3 points and peaks a level early. Of the three delivery systems, xAI's own is the only one where paying for more reasoning reliably returns more correctness.

2. 92.8% is the best result this suite has seen from Grok 4.6. Higher than any level of either other harness, and 14.5 points above the raw API at the same xhigh setting. pass@3 reaches 95.7%, which is 22 of 23 tasks solved at least once.

3. A task nothing had ever solved fell. *sqlglot-canonicalize-internal-names* went unsolved in every run of Report No. 15, by both the API and Cursor, at every setting. Grok Build solves it 2 of 3 times at medium. *pennylane-trotter-fragmented*, also never solved by the raw API, climbs 0/3, 1/3, 2/3, 3/3 across the knob: the effort curve reproduced inside a single task.

4. Both vendor harnesses share a blind spot the minimal loop does not have. *itertools-strip-prefix* is solved by the bare API at all four levels but fails 12 of 12 Cursor attempts and 11 of 12 Grok Build attempts, and the failure is identical in both: not a missed fix but a regression, breaking an existing passing test while making the change. Richer scaffolds edit more broadly, and on this task breadth is the defect.

5. Speed and accuracy moved together. Grok Build is fastest at every level while scoring highest, finishing high-effort tasks 3.6 times faster than the raw API at the same setting. Its own time curve is nearly flat from medium up, so the extra tokens at xhigh buy accuracy without buying latency.

Table 4. Task-level consistency inside Grok Build (three attempts each)

Effort	Solved every attempt	Mixed	Never solved
low	18/23	3	2
medium	18/23	4	1
high	19/23	3	1
xhigh	**21/23**	**1**	**1**

Consistency tightens as effort rises: 21 of 23 tasks solved on every attempt at xhigh, with a single mixed task. Only *itertools-strip-prefix* is never solved above low.

Caveats

Grok Build is xAI's harness running xAI's model, so this study cannot separate scaffold quality from first-party advantage: prompt tuning, serving, and model variant all sit on the same side of the comparison. There is direct evidence of the last one. VulcanBench requested `grok-4.6` and the session receipt reports `grok-`

`4.6`, but the CLI's own usage ledger names the served model `grok-4.6-build`. Whether that is a distinct checkpoint or an internal billing label is not visible from outside, and it is the same class of unknown that "Cursor Grok 4.6" raised in Report No. 15. Attempt counts differ across systems (3 vs 3 vs 1), so the bare-bones column carries the widest error bars and its per-task results turn on single runs. Costs for the bare API in Report No. 14 are lower bounds, because xAI reports reasoning tokens outside `completion_tokens` and the harness under-counted them during that sweep, so the cost comparison across systems is not clean; the Grok Build figures here are fully measured. Grok Build's advantage over Cursor is inside the error bars at every level, and only its margin over the bare API at high and xhigh clears them. The integrity audit certifies that no run reached the web or the benchmark's own files; it cannot certify what was in the model's training data.

Method

VulcanBench v3: 23 tasks from real merged post-cutoff PRs (Python 9, TypeScript 4, Rust 4, Go 3, JavaScript 3). All three systems are graded by the same deterministic hidden tests in Docker; model judges disabled. System A ran 2026-08-12, System B 2026-08-15/16, System C 2026-08-17/18 with grok 1.0.5, three attempts per task at each of four levels. Grok Build runs used a custom kernel sandbox profile (Seatbelt) that permits workspace writes while denying reads of the VulcanBench checkout, an agent workspace created outside the repository, web tools removed from the agent entirely, and cross-session memory disabled so repeats cannot recall each other. Every run carries an integrity audit of both leak channels and all 277 are clean. Effort is not taken on trust: each run records the level the CLI reports it actually used, and all 277 match what was requested. pass@1 is the mean per-task success rate, so uneven attempt counts do not bias it. One level, xhigh, failed to start on its first attempt and was rerun in full.