research · 2026-08-27

A safety filter moved our benchmark by seven places

Plicara Research · 2026-08-27 ·

This article is about a failure mode that sits underneath a benchmark score rather than inside a model, and about the sampling discipline that makes it visible. While scoring eleven models on four hundred fifty regular-expression tasks, a response from `claude-opus-5` came back with `finish_reason` set to `content_filter`, an upstream stop reason of `refusal`, one token and no text, in answer to a prompt asking for a pattern matching strings with no quotation mark in them. On our second corpus, StructuredRegex, a regex-synthesis benchmark of natural-language descriptions paired with matching and non-matching examples, the same thing happened to 182 of 622 instances, 29.3% of everything we asked; retrying recovered 98 of them and 84 stayed blank. We are in no position to certify our own prompts harmless and the argument does not need us to, because the same prompt resent unchanged comes back answered, so whatever the classifier reacted to was not a stable property of a question about quotation marks.

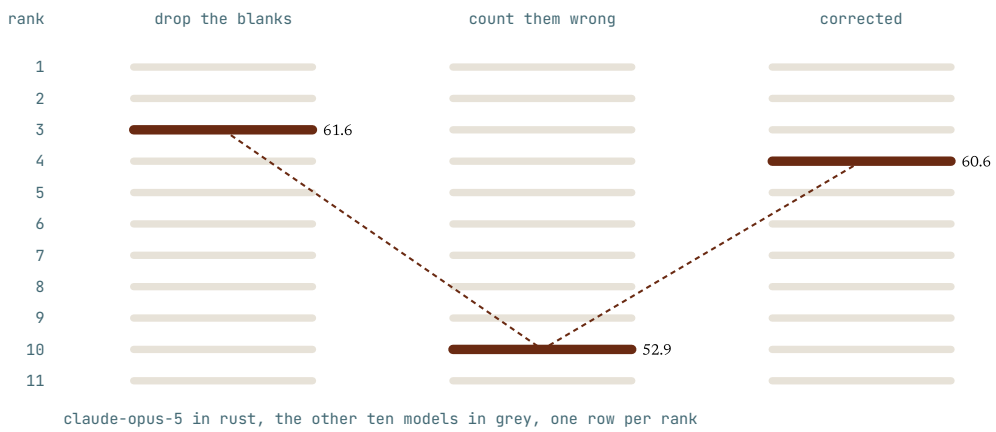
What makes this worth writing up is not that a filter fired. Filters that return empty bodies are documented behaviour: Microsoft's Foundry Models content-filtering documentation specifies the shape as contract, stating that non-streaming completions return no content when the content is filtered and the `finish_reason` is set to `content_filter`, and Anthropic documents `refusal` as a stop reason that safety classifiers return on a normal HTTP 200, with a `stop_details` object naming the policy category. What is undocumented is the stochastic, per-tier version we hit, and what our data adds is incidence, a rate, a resend-recovery fraction and a measured effect on a published score. The adjacent literature does not cover it either. Over-refusal benchmarks such as XSTest, OR-Bench and SORRY-Bench measure models declining benign prompts in visible prose, one sample each, and published Claude over-refusal rates sit in the low single digits against our 29.3%; the numbers do not conflict because they count different things, and ours never reaches the model's text at all. Work on refusal as a single direction in activation space explains why a near-threshold decision might look sampling-dependent, but it studies refusal inside the model, and a provider filter sits outside the model entirely.

The instrument we needed already exists in the evaluation literature, pointed somewhere else. Repeated sampling entered evaluation as a capability estimator with pass@k, became a statistical obligation through Miller's error-bar work, and was shown to matter for rankings

by "Do Repetitions Matter?", which found single-run leaderboards brittle enough that 10 of 12 slices invert at least one pairwise rank. All of it attributes cross-run movement to the model and to sampling. The nearest argument to ours, "Stop Comparing LLM Agents Without Disclosing the Harness", holds that harness-level variance can exceed model variance and reverse rankings, but it concerns agent scaffolds rather than provider-side filtering and offers no detection mechanism or score correction. Our contribution is to point the same instrument at the infrastructure, and the methodological ancestor for that is not in machine learning at all: internet censorship measurement has spent twenty years probing an opaque intermediary repeatedly to separate deliberate blocking from benign failure, and Clayton, Murdoch and Watson at PET 2006 established that censors make inconsistent per-request decisions, which is the coin flip we hit, described in a different medium two decades earlier.

our most relevant finding

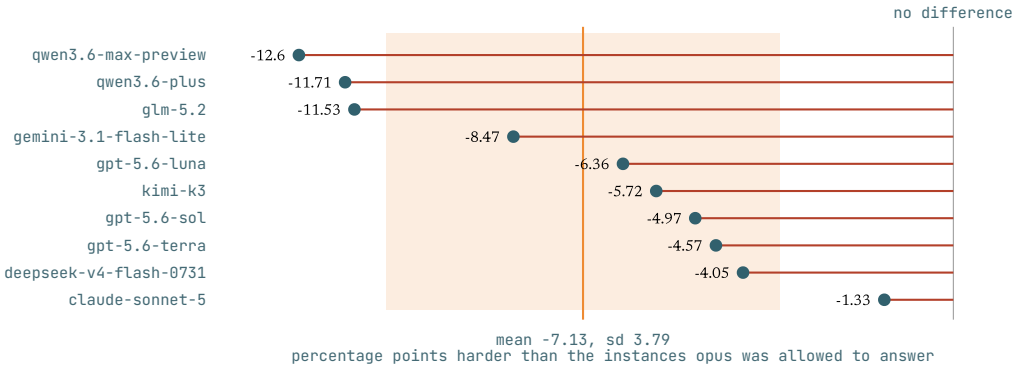
A harness that meets a blanked call has three options and no guidance: it can drop the row, score it wrong, or correct for it, and nobody documents which they chose.



The same model, the same answers, the same corpus. Only the handling of 84 blanked calls differs. Filled bar is claude-opus-5.

HANDLING	SCORE	RANK OF 11
Drop the blanked rows	61.6	3
Count them wrong	52.9	**10**
Correct for their difficulty	60.6	4

Holding everything else the same, the model, the answers and the corpus, that choice alone moves the result 8.7 points and 7 places. Neither of the first two options is defensible. Dropping the blanked rows is a complete-case analysis, and complete-case analysis is only unbiased when the missing data are missing completely at random, which these are not: asked of the other ten models, the blanked instances came out 7.13 points harder than the instances opus was allowed to answer, with a standard deviation of 3.79, a jackknifed standard error of 1.2 and a sign-flip test at p = 0.001953, with all ten models agreeing in direction.

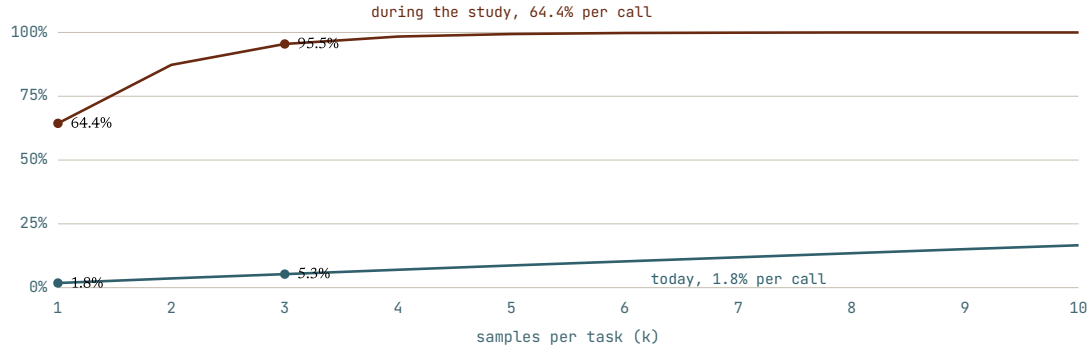


One row per model, none of them the filtered one. For each, we scored the 84 instances the filter blanked for opus and the 513 it did not, then plotted the gap between those two pass rates. A dot left of the zero line means that model also found the blanked instances harder. The orange band is the mean of the ten, plus or minus one standard deviation.

Counting the blanks wrong is worse, because it charges a model for an answer it was never allowed to give, and the corrected row is the only one of the three that is trying to measure the model at all: it imputes the blanked instances at the difficulty the other models found in them. That correction is cheap and any benchmark with more than one model can run it, since it needs no new data and no new API calls, only the instances one model lost, asked of the others, with their pass rate there compared against their pass rate elsewhere.

why k=3 caught it and k=1 would not

The filter is stochastic rather than deterministic, which is what makes repeated sampling the detection instrument: resending twenty blocked descriptions unchanged brought eight of them back answered, and in the main sweep the fifteen affected tasks produced 29 blanks across 45 attempts with ten of the fifteen answered at least once. A model that declines writes you a sentence explaining itself; this returns nothing, sometimes.



Chance that k samples of one affected task show the filter at least once, at the per-call rate we measured then and the rate it runs at now.

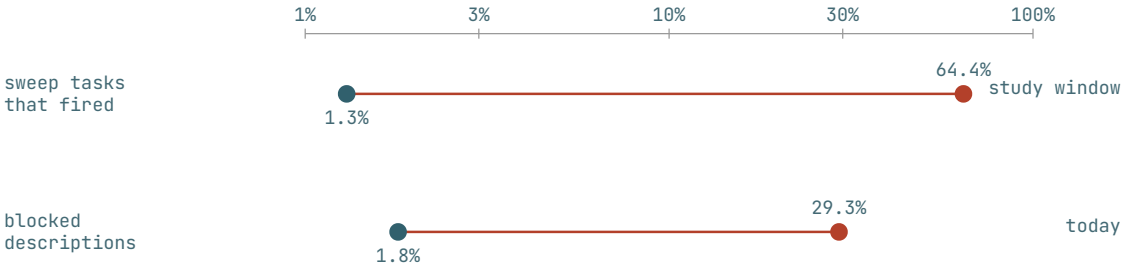
Every StructuredRegex description the filter touched, by what it did to them. Then: 70 descriptions. Now: 4.

At the per-call rate we measured during the study, one sample of an affected task caught the filter 64.4% of the time and three samples caught it 95.5%, so a single-sample run would have scored the survivors, reported a number, and never known the population had been altered underneath it.

The uncomfortable part is which way that curve moves as the filter fires less often. It now fires on 1.8% of calls against 29.3% then, and at today's rate one sample catches an affected task 1.8% of the time, three samples 5.3%, and you would need k=10 to reach 16.6%. The bias does not shrink with the rate, only the evidence does, which means a benchmark published today at k=1 is more likely to be affected and unaware than ours was. This is the same brittleness "Do Repetitions Matter?" measured when it found most of its slices inverting a pairwise rank on a single run, and it compounds with the serving-stack nondeterminism documented elsewhere, where numerical precision alone makes identical requests return different outputs. Those strands share a conclusion that has not been stated plainly: a single sample is not a measurement of a model, it is one draw from a system whose variance nobody has characterised. The practical rule is therefore not that three is enough, but that k is a detection instrument as well as a variance estimator, and the less often a fault fires the more of it you need.

it moves, and it travels with the weights

Two weeks after publication we re-ran both corpora and found the rate had collapsed without announcement, 64.4% falling to 1.3% on one corpus and 29.3% to 1.8% on the other, factors of 50 and 16.



Refusal rate on identical prompts, study window versus today. Log scale.

It did not fade evenly, though: the 16 blanks that remain fall on four of the 91 descriptions we resent, and one came back empty on all ten attempts, so the dial narrowed rather than relaxed. Pinning the same fifteen prompts to each of OpenRouter's five hosting platforms separately, 45 calls each, produced blanks on four of the five, every one carrying the same native `refusal`, so you cannot route around this by changing landlord.

45 calls per hosting platform, one square each. Blanks on 4 of 5; the clean arm is underpowered, not exempt.

One arm was clean, and it was Anthropic's own endpoint, but that is most likely luck at this sample size: at the 3.9% rate the other arms show, 45 calls return zero blanks about 17% of the time, so the honest reading is an underpowered arm rather than an exempt one. The tier asymmetry is better supported: `claude-sonnet-5` never blanked once across 150 fresh attempts at the prompts that trip its larger sibling and 398 more on the second corpus, though that arm did not finish either, because we planned 910 calls and the budget ran out partway, leaving 512 calls returning HTTP 403 without ever reaching a model.

Those are transport failures rather than blanks, and they belong in the record rather than quietly out of a denominator.

our recommendation

Log every non-answer, with the upstream stop reason attached, and learn what the codes mean. This is the whole of it: we only have an article because our harness recorded `finish_reason`, the native stop reason and the token count instead of coercing a blank into an empty string. A log line that says a call failed is not enough, because filter stops, empty bodies, transport errors and genuine model refusals need different handling and a single failure count hides all four. The harnesses most people run do not make this distinction for you: lm-evaluation-harness substitutes an empty string for a blocked generation behind a warning log, and HELM raises an empty-content error, marks it non-retriable and scores it wrong, which is precisely the 10th-place row in the table above. We caught this early, in our own draft, and only because those fields were there: an earlier version of this article merged 84 filter blanks with 25 unrelated empty-code-fence rows into one count of 109, and reported that sonnet arm's 398 successes without mentioning the 512 calls that never landed. The logs are what let us take both back.

Retry what fired before an answer existed, and never what declined after. A refusal written in prose is a model output, so resending it is just asking until you get a yes; a blank that arrives with an upstream filter stop and no text is closer to a dropped packet, and resending it measures transport luck rather than model quality.

Before reporting the survivors, test whether the lost subset was missing completely at random. Dropping filtered rows assumes MCAR, and that assumption is checkable rather than a matter of judgement: ask the other models the instances one model lost and compare their pass rate there against their pass rate elsewhere. Ours failed that test at $p = 0.001953$ in a unanimous direction, and any benchmark carrying more than one model can run the same check over data it already has.

Running $k>1$ is what makes all three possible, and not only for the error bars that are the usual argument, because a single sample cannot distinguish a model that failed from a model that was never asked.

Every number here resolves to a machine-readable export in the project's repository, and the collection scripts run against committed manifests. Two figures cannot be derived from the committed predictions, which record only the post-retry state: the 182 instances blocked on the first pass and the 98 that retrying recovered are carried from the parent study, regexeval-2026, and the export asserts they still reconcile with what the corpus can

prove, 182 minus 98 leaving the 84 blanks the predictions contain, with the build failing if they ever disagree.