PURETENSOR
INTELLIGENCE DISTILLED

The Gate That Passed by Luck

Stochastic conformance and the proof that a parameter was honoured

Paper: PT-R-2026-026 v1.0

Author: PureTensor Ltd

Date: 24 September 2026

Status: Published

Abstract

This paper describes two gates in our fleet. Each passes or fails a stochastic system on a single draw, and in each case we had read a single draw as a statement about the system. Both were measured on 22 and 23 September 2026. The first is the conformance gate on our resident local model seat, Qwen3.8-Flash-Next served by vLLM. The second is a rule for proving that a hosted API honours a reasoning-effort parameter, which we applied to xAI's grok-4.6 and grok-4.7.

In the first case, the gate's tool-calling check failed on 9 of 246 hourly runs between 12 and 23 September (3.7%, 95% Wilson interval 1.9% to 6.8%). Every failure cleared at the next run without intervention. After the resident seat was restored on 23 September, the same check failed repeatedly. We replayed the failing request 10 times. The request offered exactly one tool, `bash`. Seven replays called `bash`. Three called `calc_calculate`, a tool that had not been offered. The point estimate is 30%, with a 95% Wilson interval of 10.8% to 60.3%. The wrong name reached the client because vLLM on this seat does not constrain the emitted tool name when a call is required. The two failure rates cannot both describe one fixed process, and we did not find what separated them.

The fix was to send all conformance requests at temperature 0 instead of 0.2. All 21 runs recorded after the change that day passed the tool-calling check. The change makes the gate a repeatable measurement of the serving path. It does not remove the model's tendency to call tools it was never given at production-like temperatures, and we now record that as a known gap that consumers must defend against.

In the second case, the hosted endpoint accepted an invented request field without error. Its acceptance of `reasoning_effort: xhigh` was therefore no evidence that the setting did anything. We require every high-effort sample to clear every low-effort sample by at least 1.5 times and by at least 256 reasoning tokens. On three samples per arm, grok-4.6 passed: its lowest `xhigh` sample was 2.37 times its highest `low` sample, a gap of 4,325 tokens. grok-4.7 did not pass. Its arms did not overlap, but at their nearest points they were separated by only 1.10 times (646 tokens), even though their medians differed by 2.42 times. One benchmark comparison was released and the other remains unvalidated.

The claims that survive are limited. Both gates were random variables, and an earlier pass from a single draw is not evidence of an earlier healthy state. The specific rates, and the grok-4.7 verdict in particular, rest on small samples and could move.

1. Two Gates Built on a Single Draw

We use gates to decide whether something is fit to run. A gate must be able to see the system in a failing state, and its verdict must mean the same thing each time it is read. The second condition is easy to lose when the system under test is a sampled language model. A check that sends one request and inspects one reply looks deterministic in its code: one input, one comparison, one boolean. When the reply is sampled, the verdict is a draw from a distribution, and the check reports whichever outcome the sampler produced.

This paper describes two occasions when we had to confront that. In the first, a gate on our own serving path failed after a routine restore. We treated it as a regression at first, and only later saw that it had been failing occasionally, and paging, for eleven days before. In the second, a hosted API gave us no reliable signal about whether one of its parameters took effect, so we had to build a probe that tolerates the API's own sampling noise. We report the numbers as recorded and state which inferences they support.

2. The Resident Seat and Its Conformance Gate

The fleet runs a resident local model seat: Qwen3.8-Flash-Next with FP8 weights, served by a development build of vLLM on a two-GPU compute node, at reasoning effort `low`. Reasoning effort is a serving-side setting that governs how much hidden deliberation the model does before answering. This model has been resident since 2 September 2026. Other fleet services call it through an OpenAI-compatible chat endpoint with tools.

A conformance gate guards the seat. It was built on 30 August 2026 and runs in three places. It is the last step of the seat's warm-up, so a seat cannot become resident without passing. It runs as an hourly sensor. It also runs inside the procedure that restores the resident after an on-demand mode, in which a larger model spread across more GPUs temporarily replaces it. A failing gate raises a critical page after 5 minutes. The gate is a sensor only and never restarts anything.

Check	What it asserts	Gating
models	the model list is OpenAI-shaped and serves the expected alias	yes
reasoning_split	an exact reply lands in the content field, reasoning in its own field, no truncation	yes
tool_sweep	for every offered tool, a forced call returns a well-formed call to that tool name, JSON-object arguments, no special-token leakage, no repetition	yes
plain_prompt_with_tools	a plain prompt with the full tool list yields a call or coherent text	yes
context	a random needle placed at 60% depth of about 80% of the advertised context is returned	yes
named_tool_choice	a named tool choice between two offered tools is obeyed	no (reported as a known gap)

Two terms in the table need explanation. Special-token leakage means the model's internal control tokens appearing in the visible output. The needle is a random string buried in a long prompt that the model must retrieve.

The tool_sweep check sends one request per tool: 21 production tools plus 2 regression names that are always swept, 23 requests per run. Each request offers exactly one tool and sets

`tool_choice=required`. That setting obliges the model to return a tool call instead of prose. The prompt tells the model to call the tool now and to invent placeholder arguments. The first tool in the list is a shell tool named `bash`. The list has not changed since 2 September.

One gap was known before any of what follows. On this seat and build, a named `tool_choice` (one that names the specific tool to call) is not enforced. vLLM builds the correct grammar constraint for it, meaning a formal description of allowed output that the decoder is forced to follow, but the named-choice path never attaches that constraint. The model is free to answer in prose or call the other tool. That is why named_tool_choice is non-gating.

3. Case 1: A Red Gate After a Restore

From late evening on 22 September until about 03:00 on 23 September, the larger on-demand model replaced the resident. During that window the gate's models check failed at 00:50, 01:51 and 02:54 because the resident alias was not being served. Those failures were expected.

Time (UTC)	Event
23 Sep, about 03:00	Resident Qwen3.8-Flash-Next seat restored; restore procedure waits for a fresh gate pass
23 Sep 03:03	Gate run after restore: FAIL on tool_sweep (plus the non-gating named_tool_choice)
23 Sep 03:07	Rerun: FAIL on tool_sweep
23 Sep, between 03:07 and 03:19 (inferred)	Investigation: 10 replays of the failing request; rendered prompt inspected
23 Sep 03:19 and 03:20	Gate runs PASS (temperature 0 applied)
23 Sep 03:21	Fix committed with a red test
23 Sep 04:24 to 23:16	All 19 further hourly runs that day PASS tool_sweep

According to the incident notes, there were 5 full gate runs around the restore and tool_sweep failed in 3 of them. Every failure was on the `bash` request. Each carried the reason `wrong-name:` followed by a tool name that had never been offered: `calc_calculate`, another `calc_*` or `data_*` name, or `plan_value_steps`. The monitoring tier's time-series store samples every 4 minutes and captured only 2 failing and 2 passing runs in this window. We return to that difference in Section 9.

We replayed the gate's exact `bash` request 10 times at the gate's temperature at that point, 0.2.

Replay of the single-tool bash request	Count
Replays	10
Called bash (the only offered tool)	7
Called calc_calculate (never offered)	3
Confabulation rate, point estimate	30%
95% Wilson interval	10.8% to 60.3%

The Wilson interval is a confidence interval for a proportion that stays sensible at small counts. Its width here reflects how little ten trials can tell us.

We first checked whether the server was supplying the extra tools. It was not. vLLM's tokenisation endpoint returns the prompt exactly as the model receives it, and for this request that prompt contained only the `bash` tool. Nothing on the server injected other tools, and no

serving configuration had changed since 12 September. The model's own reasoning text showed where the names came from. It listed a tool menu that did not exist: "1. bash 2. calc_calculate 3. data_analy...". The model had confabulated a list of tools and then, some of the time, called an item from it.

The wrong name left the server because vLLM on this seat does not constrain the emitted tool name under `tool_choice=required`. The request offers one tool and demands a call, but the name field is free text as far as the decoder is concerned. The root cause is the same as the named-choice gap we already knew about. We had recorded that gap against one setting, when it applies to both.

4. What the Gate Had Already Been Saying

Our first reading was that the restore had broken something. The gate's own history did not support that reading. The monitoring tier keeps one pass or fail value per check per run, with runs about every 62 minutes. Reading it back gave the following.

Window	Runs with a tool_sweep result	tool_sweep failures	Rate
12 Sep 00:00 to 23 Sep 00:00	246	9	3.7% (95% Wilson 1.9% to 6.8%)
23 Sep, before the fix (03:03, 03:07)	2	2	
23 Sep, after the fix (03:19 to 23:16)	21	0	

The nine historical failures fell at 12 Sep 22:44, 15 Sep 23:49, 16 Sep 00:49, 17 Sep 06:04, 17 Sep 17:26, 18 Sep 00:45, 20 Sep 00:06, 20 Sep 10:28 and 21 Sep 02:10. Each cleared at the next hourly run with no intervention. The store keeps the verdict but not the failure reason. We therefore cannot confirm that these nine were the same bash wrong-name failure, only that some request in the 23-request sweep failed.

The other gating checks were quiet over the same window. context, reasoning_split and plain_prompt_with_tools recorded no failures. models failed 9 times, all during periods when the seat was unavailable: 12 Sep 09:55 to 12:01, 13 Sep 12:16 and 19:16, a whole-fleet reboot on 22 Sep 11:08, and the on-demand mode on 23 Sep from 00:50 to 02:54. named_tool_choice failed on essentially every run, as expected.

The critical page fired 13 times between 12 and 23 September. Eight firings coincide with the isolated tool_sweep failures, with durations of 55, 115, 50, 50, 50, 40, 15 and 15 minutes. The 115-minute firing spans the two consecutive failures of 15 and 16 September. The other 5 were models firings during seat unavailability. Each tool_sweep firing resolved at the next run and was not investigated as a property of the gate until 23 September. A page that clears itself within the hour invites being read as a transient, and that is how we read these.

The two rates cannot be reconciled as one process. If the per-run failure probability were the historical 3.7%, the chance of 3 or more failures in 10 replays would be 0.48%, and of 3 or more in 5 runs 0.046%. The historical 3.7% covers a whole 23-request sweep, so the bash request alone must fail at that rate or lower, which makes both post-restore observations less likely still. In the other direction, a 30% per-run rate cannot produce 9 failures in 246 hourly runs. The confabulation rate after the restore was higher than across the preceding eleven days, and we did not find the reason.

Both regimes were stochastic. Before the restore the gate was a random verdict that came up red rarely, about 1 run in 27. After the restore it was a random verdict that came up red often. Neither the reds nor the greens described the serving path.

5. The Fix and What It Does Not Buy

The fix changes one parameter. All six conformance checks now send their requests at `temperature: 0`, where they previously used 0.2. A red test (one written to fail against the old behaviour) asserts that the conformance request body carries temperature 0. We loosened no threshold and removed no check.

The notes record that 3 of 3 live runs passed 5 of 5 gating checks immediately after the change. The store captured 2 of those, and all 21 runs recorded on 23 September after the fix passed tool_sweep.

At temperature 0 the gate measures serving conformance deterministically: whether the serving stack produces a well-formed call for each tool. A red now means that something in serving changed. It no longer means that the sampler landed badly. That is the right property for a gate that decides whether a seat may become resident.

The change also has a cost, and we recorded it in the fix itself. At production-like temperatures the resident can still call a tool it was never given, and the gate is now built so that it cannot see this. Consumers must validate returned tool names against the tools they offered. The proper remedy is server-side name enforcement, using grammar-constrained output for both `required` and named `tool_choice`. That work is not done. We have recorded the confabulation as a known seat gap so that the change does not hide it. The separate half-hourly correctness probe on the same seat still samples at 0.2. Only the conformance gate was made deterministic.

6. Case 2: Proving a Hosted API Honours an Effort Parameter

On 22 September we retro-scored 55 runs of our internal frontier benchmark (94 original items across ten dimensions, every verdict produced by code, no language-model judge) against a set of validity gates. The runs date from 15 August to 21 September. Fifty predated the serving preflight and skipped it, and 5 failed it. The preflight is an open-source checker, which we vendor, that inspects an endpoint before a scored run. Two of the five failures were grok-4.6 at `xhigh` and grok-4.7 at `xhigh`, both over the public xAI API. The other three were one other hosted model at three effort levels. None of the 55 runs had been pre-registered.

Both xAI runs were blocked by one preflight finding: the endpoint accepts an invented request field without error. An API that silently accepts fields it does not recognise gives no information by accepting `reasoning_effort: xhigh`. The request would succeed whether the parameter was honoured or ignored. The benchmark score difference between grok-4.6 and grok-4.7 was therefore unvalidated.

We set the rule for releasing such a run before running any probe. It landed at 19:05 UTC on 22 September. The invented-field finding stays blocking by default. A run is released only when a finding consisting solely of that item is backed by two behavioural probes. Each probe is bound to the endpoint, model and effort string it was run against.

1. Effort probe. Three samples at `low` and three at the requested effort, on the same fixed multi-step arithmetic prompt, at `temperature: 0` and `max_tokens: 8192`. The parameter counts as honoured if and only if min(requested) is at least 1.5 times max(low) and min(requested) minus max(low) is at least 256 tokens. The measure is the API's reported reasoning tokens. A response that reports only total completion tokens is inconclusive and never a pass, because verbosity is not reasoning.
2. max_tokens probe. No effort field, `max_tokens: 8`, and a prompt that demands an 800-word essay. The cap counts as honoured if completion tokens are at most 8 and the

finish reason is `length`.

The rule compares extremes, not averages. It asks whether every draw from the high arm clears every draw from the low arm by a margin. A benchmark run takes one draw per item, so it could land on either tail of the distribution, and separated medians are not enough to protect it. The tests encode this: a dead knob with equal spend is not honoured; a small absolute gain of 10 versus 30 tokens is not proof; overlapping samples whose medians separate are not proof; a missing reasoning-token field is inconclusive; a server error is not honoured. Twenty-one tests cover the probes, and the benchmark suite passed 98 of 98. A run forced through with a warn-only override records the override, fails validation and is marked UNVALIDATED on the leaderboard. Blocked runs and runs that skipped the preflight are marked the same way.

7. The Probe Results

The probes ran on the xAI API on the evening of 22 September. The grok-4.6 effort probe completed at 19:17:30 and its max_tokens probe at 19:18:19. The grok-4.7 effort probe completed at 19:27:58 and its max_tokens probe at 19:29:19. Reasoning tokens per sample, in the order sampled:

Model	Arm	Sample 1	Sample 2	Sample 3	Min	Median	Max	Mean
grok-4.6	low	3,159	2,088	2,031	2,031	2,088	3,159	2,426.0
grok-4.6	xhigh	13,366	7,484	8,758	7,484	8,758	13,366	9,869.3
grok-4.7	low	4,691	3,802	6,688	3,802	4,691	6,688	5,060.3
grok-4.7	xhigh	11,829	11,350	7,334	7,334	11,350	11,829	10,171.0

Applying the rule:

Model	min(xhigh)	1.5 x max(low)	Ratio min(xhigh)/max(low)	Gap min(xhigh) - max(low)	Median ratio	Verdict
grok-4.6	7,484	4,738.5	2.37	4,325	4.19	PROVEN (both conditions met)
grok-4.7	7,334	10,032	1.10	646	2.42	NOT PROVEN (fails the 1.5x condition; passes the 256 gap)

Model	max_tokens probe: completion tokens	Finish reason	Verdict
grok-4.6	8	length	enforced
grok-4.7	8	length	enforced

grok-4.6 clears both conditions by a wide margin, and its benchmark lane is now validated. grok-4.7's block stands. Its arms do not overlap, since the highest `low` sample (6,688) sits below the lowest `xhigh` sample (7,334). The nearest points are separated by 1.10 times, well short of the required 1.5. Our internal write-ups on the day described this result as "ranges overlap". The raw samples show that description is wrong. The looseness comes from the rule's own documentation, which calls the combined condition "non-overlapping". The verdict fails on the separation margin, not on overlap.

The grok-4.7 verdict rests on one sample. Its medians differ by 2.42 times, and it fails only because one `low` sample came within 1.1 times of one `xhigh` sample. With three samples per

arm, one outlier can decide the outcome. We accept that consequence of the design, because the case the rule guards against is a single benchmark draw landing on exactly such a tail. No grok-4.7 conclusion from the benchmark will be drawn until the model is re-probed. The choice of which model to use for work rested on a separate latency battery, not on the benchmark, and was unaffected.

Three further observations come out of the samples. First, grok-4.7 spent more at `low` than grok-4.6 did. The medians were 4,691 and 2,088, and grok-4.7's lowest `low` sample (3,802) exceeds grok-4.6's highest (3,159). A higher floor at `low` narrows the room for separation, which is part of why the same rule gives different verdicts for the two models.

Second, temperature 0 did not make the hosted API deterministic. At fixed prompt, model, effort and temperature 0, the spread within an arm ran from 1,128 tokens (grok-4.6 `low`) to 5,882 (grok-4.6 `xhigh`). The highest sample in an arm was up to 1.79 times the lowest (grok-4.6 `xhigh`, 13,366 against 7,484), and 1.76 times for grok-4.7 `low` (6,688 against 3,802). In Case 1, temperature 0 made our own seat's verdict repeatable. On this API the client cannot remove the sampling, so the probe has to absorb it.

Third, four of the six `xhigh` samples reported more reasoning tokens than the probe's `max_tokens` of 8,192: 13,366, 8,758, 11,829 and 11,350. The max_tokens probe, sent without an effort field, stopped at 8 tokens. This suggests that on this API the `max_tokens` cap does not bound reported reasoning tokens when an effort level is set. We ran one probe per model and no direct experiment, so this is an observation, not a finding.

8. The Shared Shape

Both cases involve a gate that passes or fails a stochastic system on a single draw. In Case 1 the gate looked deterministic but sat over a sampled model. We removed the sampling from the gate and moved the stochastic behaviour into a documented gap that consumers must handle. In Case 2 the parameter's effect is visible only through sampled outputs, and the client cannot make the API deterministic. We sampled several times per arm and required separation of the extremes, not of the averages.

The doctrine we now apply to both is short. Before calling a gate failure a regression, rerun it several times and check whether the gate itself is deterministic. An earlier pass from a single draw is not evidence of an earlier healthy state. The converse also holds: a failure that clears itself at the next run is not evidence of a transient fault. It may be a draw from a gate that was never measuring what it appeared to measure.

9. What These Numbers Will Not Carry

The post-restore evidence in Case 1 is small. It consists of ten replays of one request, three of them wrong, and five gate runs, three of them failed, according to the incident notes. The 95% interval on 3 of 10 runs from 10.8% to 60.3%. The notes summarise the post-restore failure rate as "about 30 to 60%". That is a range assembled from two small samples, not a measured rate, and we do not claim it.

The incident notes and the time-series store also differ in their counts. The notes record 5 runs around the restore with 3 tool_sweep failures, and 3 of 3 passes after the fix. The store, sampling every 4 minutes, captured 2 failures (03:03 and 03:07) and 2 passes (03:19 and 03:20). Runs seconds apart can fall between samples, so the store does not contradict the notes, but only 2 failures and 2 passes are independently confirmed.

The historical rate of 9 in 246 (3.7%) covers the whole 23-request sweep. The store keeps verdicts, not reasons, so the nine historical failures cannot be attributed to the `bash` request or

to name confabulation. They are consistent with it and nothing more.

We did not identify what made the restored seat confabulate more often. The restore itself, cache state and chance all remain open. The claim that survives is that the gate was stochastic in both regimes. No specific failure probability for either regime survives.

The title of this paper needs a qualification. The notes described the earlier passes as "single lucky draws". The kind of claim is supported: each pass was a draw. The degree is not. Before the restore the gate passed about 96% of hourly runs, so most earlier passes were the likely outcome of a random gate, not rare lucky ones. What was lucky was treating those passes as evidence.

The post-fix passes (3 of 3 per the notes, 21 recorded that day) show that the gate is now repeatable on this seat and this build. They do not show that temperature-0 output is identical across restarts or across builds. We have not tested either.

Case 2 rests on one probe per model, three samples per arm, one evening and one endpoint. The grok-4.7 verdict hinges on a single `low` sample, and a re-probe could flip either verdict. The token counts are the API's reported reasoning tokens, not observed computation, and we cannot verify them independently. The observation that reasoning tokens exceeded `max_tokens` is incidental and untested. The 1.5 times and 256-token thresholds are choices, not the output of a power analysis.

The following claims survive. The conformance gate's tool-calling check was a random verdict before and after the restore. The restored seat called a never-offered tool in 3 of 10 replays of a single-tool request. vLLM on this seat does not constrain tool names under `tool_choice=required`. The rule separated grok-4.6's effort arms on 22 September and did not separate grok-4.7's. We make no quality or ranking claim about grok-4.6 or grok-4.7, and none is supported. "Not proven" means our test could not separate the arms by the required margin. It does not mean the parameter is ignored.

10. What We Changed, and What Comes Next

The conformance gate now runs at temperature 0, pinned by a test, with every check intact. Tool-name confabulation on the resident seat is recorded as a known gap. Consumers of the seat are expected to reject tool calls whose names they did not offer. The half-hourly correctness probe still samples at 0.2 and remains the place where stochastic behaviour is observed.

On the benchmark side, the grok-4.6 lane is released and the grok-4.7 lane stays UNVALIDATED until it is re-probed under the same rule. The next pieces of work on the local seat are server-side name enforcement for both `required` and named `tool_choice`, and adding failure reasons to the gate's recorded metrics. Without failure reasons, a future history like the one in Section 4 would again tell us that something failed without telling us what.

PureTensor operates a sovereign AI infrastructure fleet (on-premises GPU compute, storage, and serving) run day-to-day by autonomous agents under human direction. This paper is PT-R-2026-026. The hosted-API effort probes were run on 22 September 2026 between 19:17 and 19:29 UTC. The resident-seat conformance incident was measured on 23 September 2026 between about 03:00 and 03:21 UTC. The supporting gate history covers 12 to 23 September 2026. Raw artefacts, including probe samples, replay outputs and the gate's recorded results, are retained.