PURETENSOR
INTELLIGENCE DISTILLED

The Fix That Could Not Run

A Patch in a Module the Engine Never Loads, a NaN Argmax, and a Liveness Gate That Stayed Green

Paper: PT-R-2026-048 v1.0

Author: PureTensor Ltd

Date: 11 August 2026

Status: Published

Abstract

On 5 August 2026 we cut a speculative-decoding engine over to production: DeepSeek-V4-Flash, served by a community fork of vLLM across two workstation Blackwell GPUs, with a drafter that proposes three tokens ahead for the main model to check. Within 38 hours the engine process died four times, each time on a CUDA device-side assert (an out-of-range index caught on the GPU) triggered by a token with id 129,280, exactly the vocabulary size. Each death exited with status 0, so restart-on-failure never fired.

We wrote two patches; a 60-request burst passed after the first and a 40-request smoke after the second. Both patches sat in a model-runner module the running engine never imports, and the process we tested had started two minutes before the first edit. The verification could not have failed. The mechanism we found on 6 August, which produces exactly this id, was elsewhere: in the Triton version installed in our environment, when a row of logits contains NaN, the argmax in the target-side rejection sampler lets NaN lose every comparison, and in the padded final block the lowest masked index is 129,280. The drafter's torch argmax orders NaN the other way, so clamping draft tokens could never have helped.

After that fix, our survival gate (same start time, no restarts, no out-of-range log line) would have had nothing to flag, though no evaluation is recorded. On 11 August the same engine, at 6.6 hours of uptime, returned clean answers on 0 of 20 requests of a short exact-reply probe, the worst case for this defect, sent with a 17 KB system prompt plus tools, the shape our agent harnesses use. The engine log showed the drafter proposing tokens after the stop token. A fresh engine passed 25 of 25 on a stop-boundary probe, and a controlled A/B scored 124/125 with the drafter on and 125/125 off, inside noise. Garbled output appeared only with the drafter on, on the old engine; we did not isolate whether drafter, engine age or both are needed.

We kept the drafter on, because it decodes about 1.8 times faster, and replaced the survival gate with a correctness canary and an age-based recycle. Two rules: verify a guard by reading the loaded module's source in the running process, and never let a gate that counts restarts stand in for one that checks answers.

1. The Engine and the Question

We serve DeepSeek-V4-Flash (NVFP4 weights, 256K context) with a community fork of vLLM, tensor-parallel across two RTX PRO 6000 Blackwell GPUs on one compute node. Speculative decoding uses a drafter with k=3: a small model proposes three tokens, and the main model verifies them in one forward pass, keeping the prefix it agrees with. The engine went into production on 5 August 2026 serving our agent harnesses, which send OpenAI-compatible and Anthropic-compatible requests, many with structured output (a JSON schema enforced by a grammar state machine) or tool definitions. Local changes meant editing the installed fork. An earlier post on false-green monitoring mentioned this episode briefly; this paper gives the mechanism and what followed.

2. Four Deaths in 38 Hours

All times are UTC and come from same-day reports; the engine journal is no longer retained.

#	When	What we know of the trigger	Recovery
1	5 Aug, about 06:31	structured-output request from a coding-agent harness, about 35 min after cutover	down until a manual restart (duration not recorded)
2	5 Aug, 11:38:39	same token id, same grammar failure, same device assert	restarted by systemd at 11:38:55; about 80 s down including model reload
3	6 Aug, 03:51	Anthropic-style request with a large agent system prompt and tools; grammar rejected tokens 33, 3011 and 129280	about 80 s
4	6 Aug, 20:10:36	the same token id; both GPUs logged an Xid 43 as a consequence	about 80 s

Table 1. The four engine deaths. Intervals between them were 5 h 07 min, 16 h 13 min and 16 h 19 min; the span from first to fourth is 37 h 39 min.

Cutover to the fourth death is 38.2 hours, about one death per 9.5 hours.

The failure chain was the same every time. A token with id 129,280 reached verification; the vocabulary runs from 0 to 129,279, so this is the first slot of the padded tail of the output head. The grammar state machine could not advance on it, but the id was already queued as input to the next forward pass. The embedding lookup hit a device-side assert on both tensor-parallel workers, the engine core raised its dead-engine error, and the API server exited with status 0.

That status is the first trap. Systemd recorded Result=success, and the unit used Restart=on-failure, which acts only on a non-zero exit or a signal; a clean exit after a fatal engine error looks like a service that finished its work. We already knew this: on 26 July the same behaviour had taken another seat down three times in 75 minutes, and we had changed three vLLM units to Restart=always. The new unit was not covered, because the rule lived in a note, not in a check. Restart=always with three starts permitted per 600 seconds, added on 5 August, worked: deaths 2 to 4 each healed in about 80 seconds.

3. Two Patches That Could Not Run

On 5 August we wrote a first patch (file written 06:40:18) that, immediately after the drafter proposes tokens, replaces any draft id at or above the vocabulary size with 0. It handled the tensor return type only. After death 2 we wrote a second (file written 17:10:12) that also handled the list-of-lists return type, believing the first had been skipped on the list path, and recorded that as the reason death 2 happened.

Both patches were written into the installed file for the older (V1) model runner. The engine logs "Using V2 Model Runner" at start-up and executes a different file; the V1 file is never imported.

Both patches were dead code, and the explanation for death 2 was never tested: a file that does not run cannot explain a crash.

The verification could not have failed, for three reasons.

Reason the check could not go red	Detail
Wrong module	the guard sat in a file the running engine never loads
Process older than the edit	the process under test started about 06:38 and the first patch was written at 06:40:18; Python does not reload an imported source file
Burst too small for a rare fault	60 requests (structured output, tools and plain chat) passed 60/60 with no crash signature; at roughly one death per 9.5 hours of service, an unpatched engine could well have passed too (the burst duration is not recorded)

Table 2. Why the first verification was unfalsifiable. The 40-request smoke after the second patch was, the report said, not proof.

The same misreading appeared twice more: a recovery report said the recovered process "already loaded the patched source" after checking the V1 file, and a report on death 3 built a theory about the clamp's tensor handling although the clamp was not executing. Both times the check was whether the patch was in the file.

4. The Real Mechanism: A NaN Row Resolves to the Vocabulary Size

On 6 August a root-cause pass found that the verifier, not the drafter, produced the bad id. The target-side rejection sampler is a Triton kernel in the V2 runner. It computes a block-wise argmax over the logits: the vocabulary of 129,280 is split into blocks of 1,024, which makes 127 blocks. The last block holds 256 real lanes (ids 129,024 to 129,279) and 768 lanes masked to minus infinity, the first of which is id 129,280. In the Triton version installed in our environment, the argmax combine function accepts the first operand if it is strictly greater, or if the two are equal and its index is lower. NaN compares false with everything, so a NaN lane never wins a comparison. The fault is the kernel's use of the combine function with padded lanes, not a claim about the library.

If a row contains NaN, the NaN lanes therefore lose to the minus-infinity lanes in the last block, and the lowest index among those is 129,280. The output is forced, not random. The drafter's cross-block reduction is a torch argmax which, in the installed version, treats NaN as the maximum and returns an in-range NaN lane. That asymmetry is why only the verifier emitted the out-of-vocabulary id, and why clamping draft ids could never have stopped it; our earlier notes, which blamed the drafter, were wrong.

We reproduced this on 6 August against the running environment's actual rejection-sampling function: one request, one draft token, and a bonus-position row whose logits we varied.

Bonus-row logits	Unpatched	With the fix
all NaN, temperature 0.0	129280 (out of range)	0
all NaN, temperature 0.7	129280 (out of range)	0
all minus infinity (grammar allows nothing)	0	not recorded
normal row (control)	the correct token	not recorded
NaN plus one finite logit	not recorded	the true token

Table 3. Deterministic red/green on the real code path. These cells are taken from the patch header and the same-day report; the probe script's per-cell output was not retained, and the

patched-engine cells for the last two unpatched rows were never written down, so we treat the table as summary-level.

The fix, live at about 21:16 on 6 August, has three parts. A one-line map from NaN to minus infinity runs before the block argmax at both argmax sites in the rejection-sampler kernels; it needs no host synchronisation and leaves non-NaN rows untouched. A backstop after sampling in the V2 runner fills any sampled id at or above the vocabulary size with 0. And the engine's NaN counter was switched on, so a recurrence could be told apart from a different producer.

We defined four gates for it: the red/green test above (passed); a loaded-source proof, in which the journal shows "Using V2 Model Runner" and the source text of the loaded function, read from the running process, contains the patch marker (passed at 21:18); forty-eight hours in service under structured-output traffic with no restarts and no out-of-range log lines, which could not close before 21:16:38 on 8 August (the window opened at 21:16:38 on 6 August); and the NaN counter as precursor telemetry. The loaded-source proof is the rule we kept: read the source of the loaded module in the running process, plus a log line that proves which implementation is active, never the patched file.

5. A Gate That Counted the Wrong Thing

The time-in-service gate was a survival measure: the same engine start time, no restarts, and no journal line matching the out-of-range signature. Its author had written that 48 clean hours of near-idle service would be a weak green; the warning was not acted on for two gate cycles.

We found no record of the gate being evaluated and passed, so we claim no number of hours of green. On 7 August the engine start time was unchanged and the restart count was 0; we have no record for 7 to 11 August. Reports record zero out-of-range lines from the fix going live at about 21:16 on 6 August until the engine was changed on 11 August (about 99 to 105 hours, against one death per 9.5 hours before), with the caveat that the journal is gone. An engine start occurred at 00:18 on 11 August, cause not recorded; the gate's one-start-time criterion could not have tolerated it, but no evaluation after it is recorded.

On 11 August a coding-agent harness was reported as corrupted; its model backend was returning fluent but wrong text, and a session had died on an HTTP 500 at 06:42. That morning, before 06:57, with the drafter on and the engine at about 6.5 to 6.6 hours of uptime, we sent twenty requests per shape asking for a fixed short exact reply.

Request shape	Clean out of 20
bare prompt, no tools	20
17 KB system prompt, no tools	17 (3 garbled)
17 KB system prompt plus tools	0 (19 garbled, 1 HTTP 500)

Table 4. Clean replies by request shape on the 6.6 hour engine, drafter on. The last row is the only shape every one of our agent harnesses sends.

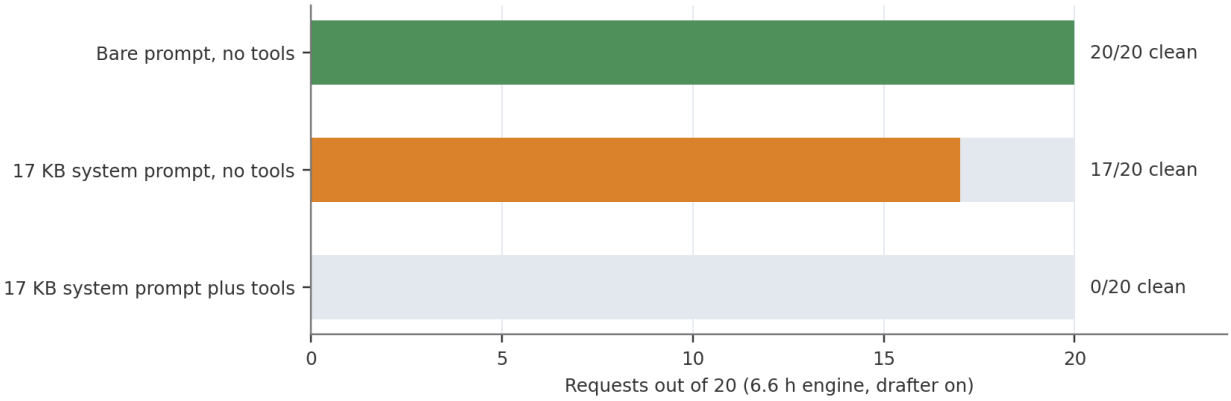


Figure 1. Clean replies out of 20 by request shape on the 6.6 hour engine. Values are in Table 4.

Garbled output contained the raw begin-of-sentence special token and fragments of unrelated code. The engine log showed the drafter proposing tokens after the stop token had been accepted, and the grammar matcher logging that it "has terminated after accepting the stop token, but is trying to accept new token". Our reading is that on plain requests these tokens leak into the reply, and on structured or tool requests the request is terminated and the caller receives a 500. The 24 hours before we turned the drafter off held 212 such warnings and 4 FSM failures, spanning more than one engine because of the 00:18 start.

This is a different defect from the deaths: no crash, HTTP 200, wrong text. The process never restarted and the out-of-range count stayed at zero. The gate measured availability and one crash string, and could not go red on this failure.

6. How Bad Was It? The Headline Was the Worst Case

At 06:57:38 we turned the drafter off with a switch in the serve script (off by default, on with an environment variable). Across the three shapes the engine then scored 60/60 clean, with no post-stop warnings, FSM failures or out-of-range lines. The throughput cost was 193.2 down to 109.3 tokens per second (a factor of 1.77).

The operator reported that the model seemed fine in ordinary use, so a controlled A/B followed: one restart between arms, 125 graded calls per arm (25 per task), and the 17 KB system prompt on every call. The off arm ran earlier, on the 06:57 engine (at most 1.4 hours old; its start is not recorded); the on arm ran from 08:21 to 08:34 on an engine at most 13 minutes old. Grading was mechanical: exact match, parsed integer sequence, executed Python, parsed tool-call JSON.

	Drafter on	Drafter off
correct	124/125 (99.2%, Wilson 95% interval 95.6 to 99.9)	125/125 (100%, interval 97.0 to 100)
silent wrong (no leak, repetition or error)	1	0
special-token leaks	0	0
HTTP errors	0	0
decode speed, bare prompt, tok/s	168.6	96.8
decode speed, agentic shape, tok/s	178.4	97.4

Table 5. Graded A/B; the drafter-on arm ran on a young engine (at most 13 minutes), the off arm on one at most 1.4 hours old. The speed ratios are 1.74 (bare prompt) and 1.83 (agentic shape).

The single silent failure was a wrong answer to an integer multiplication of two two-digit numbers; the off arm answered it correctly 25 of 25 times. One event in 125 has a 95% interval of 0.1% to 4.4%, so it cannot be separated from ordinary small-model arithmetic error. The A/B shows no defect on a young engine; it does not show the defect is absent.

The two measurements were taken on engines of different ages.

Engine life, 11 Aug	Drafter	Uptime at end	Post-stop warnings	Stop-boundary probe
00:18 to 06:57	on	6.6 h	94	0/20 clean
06:57 to 08:21	off	1.4 h	0	20/20 clean
08:21 to about 08:30	on	about 10 min	12	25/25 clean

Table 6. The same configuration on a long-lived and a fresh engine. The probe sampled about 08:30, before the A/B on-arm ended at 08:34. The request shape behind the 20/20 and 25/25 rows is not recorded, so the 0/20 may not be compared with them like for like.

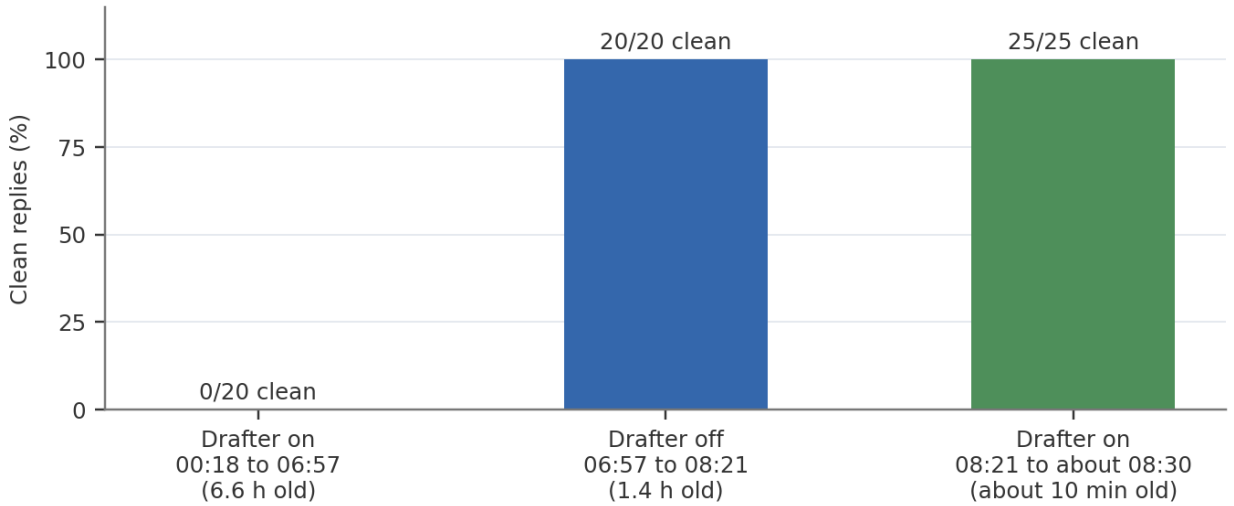


Figure 2. Stop-boundary probe results by engine life. The long-lived engine with the drafter on scored 0 of 20; the fresh engine with the drafter on scored 25 of 25. Values are in Table 6.

Accumulated cache state is a suspect (prefix caching is on); we did not test it. Three cautions apply. The 0/20 probe used a stop-boundary prompt (a few tokens, then an immediate stop), the worst case for a post-stop defect, and we first reported it as the average case. The long engine also served real traffic while the short ones mostly served benchmarks, so uptime is confounded with traffic. And the warning counts are not normalised to requests: the fresh engine logged 12 in about ten minutes (roughly 70 an hour under benchmark load), against 94 in 6.6 hours (14 an hour) and 212 in 24 hours (8.8 an hour, across more than one engine). The count marks the presence of the drafter, not the degradation.

7. What Is Still Unknown

We do not know why the logits were NaN. The hypothesis is a numerical issue in the quantised matmul or cache path on workstation Blackwell, but we have no evidence for any specific component, did not test them, and have no record of what the NaN counter read. We captured no real NaN row; the link to the synthetic row rests on the id equality and the identical failure chain. The recurrence-free period for the NaN fix (about 99 to 105 hours, Section 5) is report-level only, with no journal. The cause of the post-stop defect is open: the warning appears on every drafter-on engine, including clean fresh ones, and drafter-off was only ever

tested after a restart, so drafter and engine age are confounded.

8. What These Numbers Will Not Carry

There are four deaths and a mechanism established on a synthetic row. The NaN mechanism is consistent with deaths 3 and 4, where a guard existed on disk but not in the loaded module. Deaths 1 and 2 share the failure chain and token id, but we have no direct evidence of which kernel produced the id.

The uptime dependence rests on one long-lived engine against two short ones, with no re-probing at intervals. The A/B arms were not randomised (off arm first, on an engine at most 1.4 hours old; on arm second, on one at most 13 minutes old), and 124/125 against 125/125 cannot establish a difference.

What survives is narrower than the story. A guard in a module the engine does not load cannot be verified by exercising the engine. The argmax asymmetry produces id 129,280 from a NaN row, and the fix maps that to a valid token in the tested cases. A survival gate cannot detect wrong answers served over HTTP 200. The 0/20 figure is a worst-case probe on an old engine, not a steady-state rate.

9. What We Changed

The operator kept the drafter on, judging the speed (about 1.8 times) worth the risk. On 11 August we replaced the survival gate with a canary and a recycler.

The canary runs every 30 minutes: a stop-boundary prompt against the real 17 KB system prompt plus tools, four calls per run. It restarts the engine only if two or more of the four fail, because restarting on one failed sample would turn an assumed error rate of about 1% (one failure in 125) into a restart loop. An hourly recycler restarts the engine once its uptime passes four hours, deferring if traffic arrived in the last 90 seconds or another job holds the GPU. An alert fires if the canary itself goes quiet.

We red-proofed it: degraded to restarted to a clean probe took 67 seconds, and a synthetic GPU lease correctly blocked both probe and recycle. The old gate is retired, because the recycler deliberately breaks its one-start-time condition; its replacement checks that the probe passes and the canary timestamp advances.

10. Methods and Artefacts

Engine: a community fork of vLLM using the V2 model runner, serving DeepSeek-V4-Flash in NVFP4 with a speculative drafter at k=3, xgrammar, Triton and torch kernels, on two RTX PRO 6000 Blackwell GPUs (tensor parallelism 2), under a systemd unit with Restart=always from 5 August. Probes: a script driving the loaded rejection-sampling function with crafted batches, a second running the block argmax with and without the NaN map, request-shape probes (n=20 and n=25) and a 125-call graded A/B on 11 August. Statistics: Wilson 95% intervals from the counts, no significance tests. We retain the guard patches, probe scripts, canary script and same-day reports; journals, per-call outputs and A/B raw records were not retained.

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-2026-048. The deaths occurred on 5 and 6 August 2026 and the correctness probes and A/B were run on 11 August 2026; the patches, probe scripts, canary and same-day reports are retained, while the engine journals and per-call probe outputs are not.