PURETENSOR
INTELLIGENCE DISTILLED

Thirty-Two Tails Were Not Enough

A sliding-window cache setting that made our seat recompute 3.3 times the prompt tokens it needed

Paper: PT-R-2026-056 v1.0

Author: PureTensor Ltd

Date: 20 October 2026

Status: Published

Abstract

Our resident local model seat serves long, append-only agent prompts, which are cheap only if the engine keeps the work it has already done. On a model with sliding-window attention layers, a cached prefix can be reused only while the engine still holds a small piece of window state for its tail, and the number of tails it keeps is a setting. Ours was at its default of 32. We replayed identical multi-turn agent traffic against the seat at 32 tails and at 256 tails and compared the prefill work and latency of the two.

The workload was 320 requests built from 16 concurrent sessions of our own coding-agent transcripts, 8,804,981 prompt tokens per arm, with identical prompt lengths in both arms. At 32 tails the engine computed 1,587,200 prefill tokens; at 256 tails it computed 484,864, a reduction of 69.5%. Put the other way, the 32-tail default computed 3.27 times as much prefill as the 256-tail arm; that ratio is between the two arms, not against a theoretical minimum. The prefix hit rate rose from 82.51% to 95.04%. Time to first token (TTFT) at the 99th percentile fell from 4.23 s to 1.15 s, and requests slower than one second fell from 94 of 320 to 5. At 256 tails the uncached prompt tokens sat within about 0.1% of the workload's estimated minimum. Decode did not regress; its median rate was 13% higher.

The gain is entirely on follow-up turns. First turns of a session were unchanged (mean TTFT 0.445 s against 0.451 s), and the loss at 32 tails grew with conversation length, peaking at turns 11 to 15. The price of the larger setting is main KV-cache capacity, because tails and the main pool draw on the same GPU memory; this replay needed far less than either pool offered, so it cannot price that trade.

This is one run per arm, in a fixed order, with nothing tested between 32 and 256. Section 6 states which claims survive. The 1,102,848 cached tokens missing at 32 tails are 13.2% of the 8,368,128 that the 256-tail arm reached.

1. A Technique We Could Not Run, and a Setting We Could

The seat in question serves DeepSeek-V4.1-Flash from one SGLang process, tensor-parallel 4 and expert-parallel 4 across four RTX PRO 6000 Blackwell workstation GPUs in one node. It has 8 concurrent request slots, a 262k-token context, the model's speculative-decoding head on

(block size 5; a draft mechanism that proposes several tokens for the main model to verify), and decoder-side sliding-window-attention bounded replay on.

Its traffic is agent traffic: every turn resends the whole history and adds a tool result or a message, with idle gaps while a tool runs or a person reads. The engine's prefix cache decides how much of each prompt is paid for again.

A newly released alternative inference framework described a technique, encoder-side bounded replay for sliding-window layers, that looked relevant to long agent sessions. We could not run that framework: it shipped no kernels for our GPU architecture (we took this from its documentation and did not re-verify it for this paper). The technique could still be tested in our own engine, in principle. In our September image, per our report, encoder bounded replay refused to start together with speculative decoding, and dropping speculative decoding to test it would have changed the seat we meant to measure. Upstream SGLang lifted the restriction on 15 October, after our image was built.

So we tested the underlying mechanism through a direct setting instead. With everything else fixed, how many sliding-window prefix tails does this seat need to keep for realistic multi-turn agent traffic, and what does the default cost? It does not evaluate encoder replay.

2. What a Prefix Tail Is

DeepSeek-V4.1 mixes full-attention layers with sliding-window layers, whose window is 128 tokens. A full-attention layer needs the key-value (KV) state of every earlier token; a sliding-window layer needs only the last window. The engine's radix cache, a tree that stores KV state keyed by token prefix so that later requests sharing a prefix can skip recomputing it, therefore has two things to hold for each reusable prefix: the full-attention KV, and the paged window state for the last window of that prefix. We call the second a tail.

The upstream change that lifted the restriction describes the consequence in one sentence: "A cached prefix is reusable only while its last window still has paged SWA slots (the radix match stops at a tombstoned window)." If the tail has been evicted while the full-attention KV for the prefix is still on the GPU, the engine cannot use the cached prefix beyond the point where the window state survives. It re-prefills tokens whose KV it is still holding. The same description reports this for a one-hour agentic replay at 16 concurrent sessions on the engine project's own hardware: conversations idle for 3 to 5 minutes lost their tail while their full KV stayed on the GPU, and re-prefilled 130K to 755K tokens, blocking queued work for 2 to 13 s.

The flag `--swa-prefix-tails` sets how many tails the radix cache keeps. According to our launcher comment and our report, the default in our image is 4 times the slot count, which for 8 slots is 32, and one tail costs one window plus one page, 384 tokens (128 plus 256). We could not confirm either figure from source. We did not capture the serving image's source for this paper, and the copy of SGLang installed locally does not contain the flag. What we can say is that a page size of 256 is consistent with every cached-token count in our raw results being an exact multiple of 256 (for example 7,265,280 is 28,380 times 256, and 8,368,128 is 32,688 times 256). Everything in this paper about the mechanism rests on the engine project's public description and on our own measurements, not on the code of our image.

The upstream change also says 256 tails was too few for their 64-slot, one-hour replay; ours has 8 slots, and we do not transfer their conclusion.

Our reading: sixteen live conversations compete for 32 tails, and each can need more than one (the system preamble, plus each distinct branch point). The conversations that lose are the idle ones. We did not instrument per-request tail evictions, so this is an inference from the numbers in Section 4, in particular from cold first turns being unaffected while losses grow with turn count. It is not a traced mechanism.

3. Replaying Real Agent Sessions

Workload. We took 24 real multi-turn coding-agent transcripts from our own sessions and converted them into chat sessions. Each session received a shared agent-instructions preamble (three instruction files concatenated, about 13.8k tokens by our estimate, identical across sessions), followed by its own history, with tool results capped at 12,000 characters each and tool-call inputs at 2,000. Each session contributes 20 requests, one at every assistant-turn boundary, where the prompt is the history up to that boundary. Only sessions with at least 20 such turns were kept, and 24 qualified.

The replay used the first 16 of the 24, run concurrently. Arrivals were staggered uniformly at random over 60 s, and each session sent its turns in order. Think time between a session's requests was seeded per session: 70% of gaps were uniform 2 to 10 s (a tool pause) and 30% were uniform 30 to 150 s (a person pausing). Idle sessions therefore compete with live ones for tails. Every request forced 128 output tokens (end-of-sequence ignored), at temperature 0, with thinking mode off and streaming on.

That gives 320 requests per arm (16 sessions times 20 turns) and 8,804,981 prompt tokens per arm. Per-request prompt lengths are identical across the arms, which we checked request by request. Prompts grow along the conversation: the mean over the 16 sessions was 15,193 tokens at turn 0 (range 14,006 to 24,464) and 37,816 at turn 19 (range 31,428 to 44,603), with a mean of 27,516 tokens over all 320 requests.

Arms. Arm A ran 32 tails, the default, on the process that was already running. Arm B ran 256 tails, applied through a service override and a restart; the seat was ready after 300 s, per our report. The prefix cache was flushed before each arm (a flush call, then a 3 s wait), and both arms drew identical seeded schedules.

Measurements. We read token counters from the engine's metrics endpoint before and after each arm: prompt tokens, cached tokens, computed-prefill tokens and request count. From the client we recorded per-request TTFT (the first streamed content token) and end-to-end time. Foreign requests (the engine's request count minus the client's) were 0 in both arms, so nothing else used the seat.

Order. One run per arm, A then B. There are no repeats and no randomised order. Arm A ran from 16:35 to 16:53 UTC and arm B from 17:00 to 17:18 UTC on 20 October 2026.

4. What Changed

4.1 The headline

Table 1 gives the two arms side by side. All 640 requests completed without error.

Metric	32 tails (A)	256 tails (B)	Change
Requests ok / errors	320 / 0	320 / 0	none
Foreign requests	0	0	none
Prompt tokens	8,804,981	8,804,981	identical
Cached tokens	7,265,280	8,368,128	+1,102,848
Prefix hit rate (cached / prompt)	82.51%	95.04%	+12.5 points
Prompt tokens not served from cache	1,539,701 (17.49%)	436,853 (4.96%)	-71.6%
Prefill tokens computed (engine counter)	1,587,200	484,864	-69.5% (ratio 3.27x)
TTFT p50 (s)	0.43	0.25	-42%

Metric	32 tails (A)	256 tails (B)	Change
TTFT p90 (s)	2.59	0.56	-78%
TTFT p99 (s)	4.23	1.15	-73%
TTFT mean (s)	0.95	0.34	-64%
TTFT max (s)	5.89	2.05	
Requests with TTFT over 0.5 s	142	51	
Requests with TTFT over 1 s	94 (29.4%)	5 (1.6%)	-94.7%
Requests with TTFT over 3 s	20 (6.3%)	0	-100%
Summed end-to-end time (s)	728	404	-44.5%
Wall time (s)	1,185	1,171	-1.2%

Table 1. Counters and latency for the 32-tail and 256-tail arms on identical replayed traffic. Percentiles use the nearest-rank form over 320 requests.

Wall time barely moved because the replay is dominated by think time by design. The engine's computed-prefill counter exceeds the uncached-prompt count in both arms by about the same amount (47,499 tokens in A, 48,011 in B). That is the page and window recompute that even a cache hit pays, about 150 tokens per request. The tokens removed by the larger tail store are the same by either count: 1,102,336 fewer computed, 1,102,848 more cached.

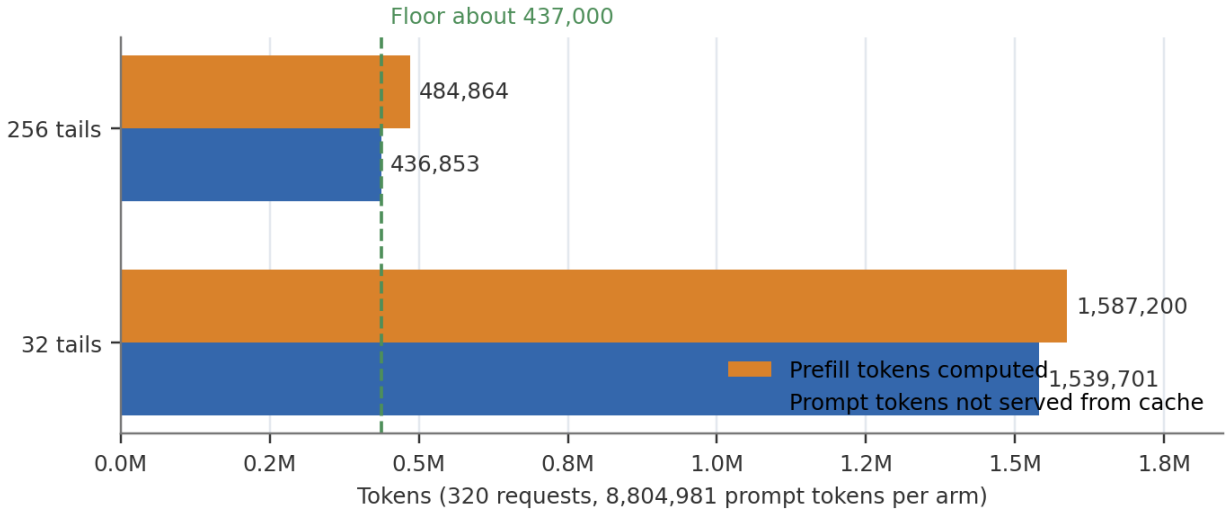


Figure 1. Prefill tokens computed and prompt tokens not served from cache, by arm, against the derived floor. At 256 tails the uncached count sits on the floor; at 32 tails it is 3.5 times higher.

4.2 How close to the minimum

For this workload we can compute a floor. The least prefill it can need is the unique tokens, meaning each session's final prompt with the shared preamble counted once, plus about one page of remainder per follow-up turn. From the raw per-request prompt lengths, the 16 final prompts sum to 605,056 tokens. Removing the 15 repeats of the preamble (about 13.8k tokens, an estimate) leaves about 398,056. The 304 follow-up turns add about 128 tokens each (38,912), for a floor of about 437,000.

Arm B's uncached-prompt count, 436,853, matches this floor to within about 0.1%. We do not claim more: the preamble length is an estimate, and an error of 50 tokens in it moves the floor by 750 tokens, 0.17%. Its computed prefill, 484,864, sits 47,896 above the floor, which is the per-request window and page recompute that every hit pays and that arm A pays too. Arm A computed 1,150,232 tokens above the floor. Of the avoidable prefill (1,587,200 minus the

floor), 256 tails removed 1,102,336, or 95.8%.

4.3 First turns against follow-up turns

Table 2 splits requests into the 16 cold first turns and the 304 warm follow-ups.

TTFT (s)	Arm A	Arm B
Turn 0 (cold session start), n=16: mean	0.445	0.451
Turn 0: median	0.207	0.192
Turns 1 to 19 (warm), n=304: mean	0.974	0.330
Turns 1 to 19: median	0.453	0.255
Turns 1 to 19: p90	2.59	0.55
Warm requests over 1 s	92 of 304 (30.3%)	3 of 304 (1.0%)
Cold requests over 1 s	2 of 16	2 of 16

Table 2. TTFT for first turns and follow-up turns; medians are the upper-middle value. Cold turns are the same in both arms; the whole difference is on follow-ups.

The cold first turns are unchanged. The two slow cold turns in each arm are the same effect, sixteen sessions starting within 60 s of each other. A first turn has no earlier prefix to lose.

4.4 The loss grows with the conversation

Table 3 groups turns in blocks of five (80 requests per block, 64 for turns 16 to 19).

Turns	Mean prompt (tokens)	Arm A mean TTFT, p90, (over 1 s)	Arm B mean TTFT, p90, (over 1 s)
1 to 5	19,208	0.55, 1.14, (11)	0.36, 0.57, (3)
6 to 10	26,700	0.81, 2.04, (24)	0.34, 0.63, (0)
11 to 15	32,265	1.50, 3.47, (36)	0.31, 0.55, (0)
16 to 19	36,064	1.06, 2.88, (21)	0.31, 0.48, (0)

Table 3. TTFT in seconds by block of turns. The prompt column is the unweighted mean of per-turn means over the block.

Arm A is worst at turns 11 to 15, with mean TTFT 1.50 s and 36 of 80 requests over a second. Arm B is flat near 0.3 s. Figure 2 shows it turn by turn: the 32-tail series climbs from 0.41 s at turn 1 to a peak of 1.92 s at turn 13 and stays about 1 s or above from turn 10 on (lowest 0.96 s, at turn 19), while the 256-tail series does not leave the range 0.25 to 0.45 s.

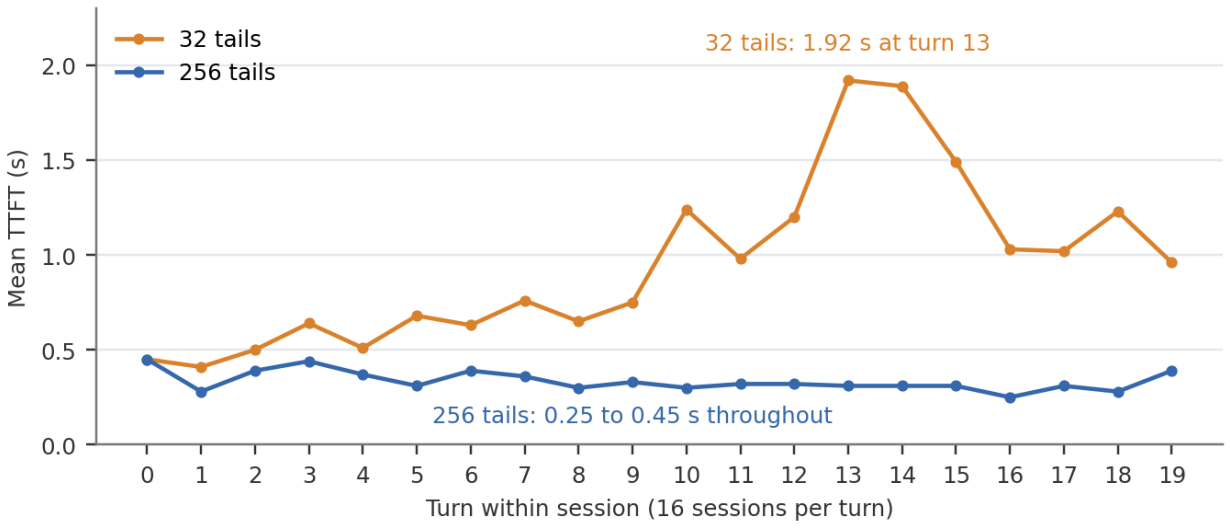


Figure 2. Mean TTFT by turn within a session, 16 sessions per point. Turn 0 is identical in both arms; the 32-tail series diverges after turn 0 and peaks at turn 13.

Grouping by prompt size (Table 4) gives the same picture. The penalty at 32 tails rises with length.

Prompt tokens	n	A mean (s), over 1 s	B mean (s), over 1 s
under 20,000	68	0.44, 3	0.35, 3
20,000 to 29,999	118	0.88, 41	0.35, 2
30,000 to 39,999	125	1.25, 47	0.31, 0
40,000 and over	9	1.48, 3	0.53, 0

Table 4. Mean TTFT and count of requests over 1 s, by prompt size, all turns. The last row has only nine requests.

Prompts grow with turn, so Tables 3 and 4 are one observation seen two ways, not two.

4.5 Where the percentiles went

The median improved by 42%, the tail far more: p90 fell by 78% and p99 by 73%.

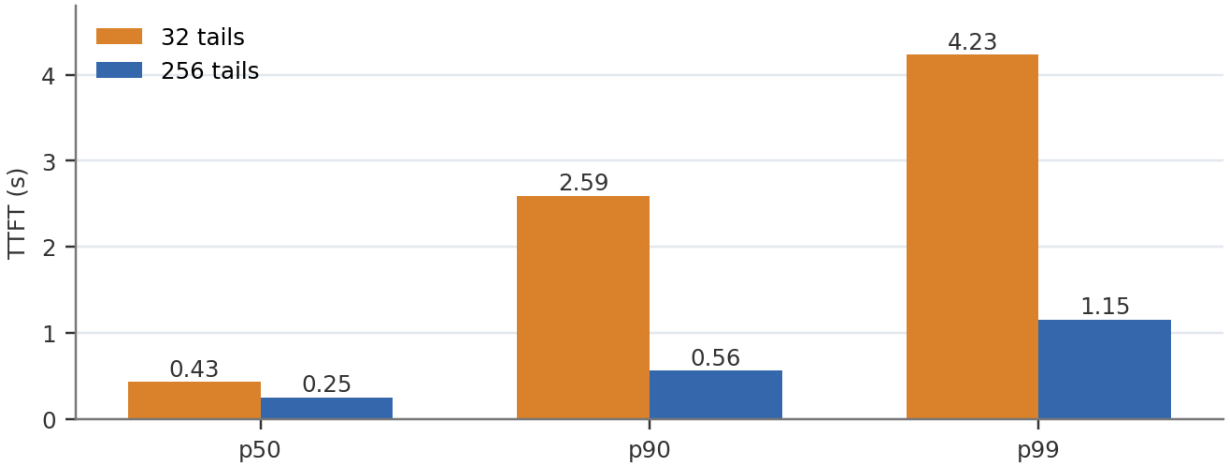


Figure 3. TTFT percentiles by arm over 320 requests each. The gain grows toward the tail of the distribution.

4.6 Decode

The median per-request decode rate, taken as 128 tokens over end-to-end time minus TTFT and recomputed from the raw records, was 138.7 tokens per second at 32 tails and 156.7 at 256 tails, 13% higher. Counting 127 tokens gives 137.7 and 155.5; our report gave 137.5 and 155.0 by a method we could not reproduce exactly. The mean over requests was 134.1 and 154.2. There is no decode regression.

5. The Price

Tails and the main KV pool come out of the same GPU memory. Raising the tail count from 32 to 256 therefore takes capacity from the pool that holds full-attention KV for running and cached requests. Our run report records the main pool falling from about 2.36 million to about 1.12 million tokens, roughly halved; we did not capture the boot logs for this paper, so treat that as unverified. At the larger setting, eight simultaneous full 262k-token requests (about 2.1 million tokens) would then no longer fit; per our report no such case appeared in the traffic we saw. What we can say from the replay itself is narrower: the replay needed far less KV than the seat had to offer, so it says nothing about main-pool pressure. A workload with more live context than the smaller pool holds would pay in evicted full-attention KV, a cost this experiment could not observe.

The change also needed a restart (ready after 300 s, per our report).

6. What These Numbers Will Not Carry

Sample size and repetition. One run per arm, A then B, no repetition, no confidence intervals. A difference of 3.3 times in computed prefill and a drop from 94 to 5 requests over a second are unlikely to be noise, but we did not test that. The TTFT percentiles over 320 requests are coarse: the p99 is the 317th value of 320.

A confound in the order. Arm A ran on a long-lived process and arm B on a freshly restarted one. Other warm state differed. The equal cold first-turn TTFT (0.445 s against 0.451 s mean) argues against a large restart effect, but it does not prove there is none.

The workload is a model of our traffic. Real transcripts reshaped into a replay: tool results truncated, outputs forced to 128 tokens, thinking off, seeded think times, and only the first 16 of 24 qualifying sessions. Hit rates include cold first turns, so a follow-up-only rate would be higher, and they are not comparable to a live-traffic hit rate.

Mechanism. We have a symptom, not a trace: no eviction logs, so tail starvation is an inference. Nothing between 32 and 256 was measured, so we do not know the knee; 256 is the value tested, not the minimum that works.

Not tested. Lower-concurrency workloads, a single-session workload, other idle-time mixes, other models, and main-pool pressure. The 32-tail default may be harmless for a workload with fewer live sessions than tails; we have no evidence either way.

Surviving claims. The large drop in re-prefill, the rise in hit rate, the collapse of the TTFT tail on follow-up turns, equal cold TTFT, no errors, no decode regression, and the fact that the setting trades against main-pool size.

A claim the counters do not support. The setting did not cost a third of prefix hits. In tokens, the 1,102,848 cached tokens missing at 32 tails are 13.2% of the 8,368,128 that arm B reached. In hit rate, 17.5% of prompt tokens went uncached against 5.0%. The figure that comes near a third is latency: 29.4% of arm A's requests, and 30.3% of its follow-up turns, waited over one second. What the data supports is that arm A recomputed 3.27 times as many prefill tokens, and that 72% of its prefill was avoidable by the floor measure above (1,150,232 of 1,587,200).

7. What We Changed

The launcher now passes a tail-count variable through to the flag, the service unit sets it to 256, and the temporary A/B override was removed. As of 20 October the seat keeps 256 tails. The next experiment is the one this one could not be: a sweep of the tail count at 64, 96 and 128 under the same replay, with repeated runs in alternating order, and a workload large enough in live context to put the main pool under pressure so that both sides of the trade are priced. The encoder-replay comparison remains unrun on our image; arm B is already at the floor on this workload.

8. Methods and Artefacts

The engine was SGLang (image built September 2026) at tensor-parallel 4 and expert-parallel 4, with the speculative-decoding head at block size 5, decoder sliding-window bounded replay on, 8 slots and a 262k context, on four RTX PRO 6000 Blackwell workstation GPUs in one node. The harness was a workload builder, a replay driver (one thread per session, a seeded generator per session, streaming requests, metrics scraping before and after, a cache flush before each arm) and a comparison script. Percentiles are nearest-rank (index int(p times n) of the sorted TTFT list). No statistical tests were run. Turn-level and block-level TTFT, the cold and warm split, decode rates, the floor and the token differences were recomputed from the raw per-request records; the headline values in Table 1 come from the result files' own summaries and were cross-checked. We retain both per-request result files (320 records each), both run logs, the scripts and the workload file, which holds transcript text and is not publishable.

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-2026-056. Both arms were measured on 20 October 2026; the per-request records and run logs are retained.