PURETENSOR
INTELLIGENCE DISTILLED

The Flat Line

Why a single throughput number cannot tell batching from queueing

Paper: *PT-R-2026-010 v1.0*
Author: *PureTensor Ltd*
Date: *11 August 2026*
Status: *Published*

Abstract

Inference benchmarks are routinely reported as a pair of numbers: single-stream tokens per second, and aggregate tokens per second at some fixed concurrency. We show that this pair is insufficient to characterise a serving configuration, because it cannot distinguish an engine that batches from an engine that queues. Both produce a plausible aggregate figure; only one of them scales.

Evaluating NVIDIA's Nemotron 3.5 Lightning 30B-A3B against our incumbent on-premises model, we found that our own production serving configuration returned an identical aggregate throughput, approximately 166 tokens per second, at every concurrency level from one to sixty-four, while 95th-percentile request latency rose from 1.55 s to 92.73 s. The engine was not slow; it was serial. A configuration constraint imposed by speculative decoding had pinned it to a single in-flight sequence, and the single-point measurement we had been quoting was structurally incapable of revealing that.

Correcting the measurement changed three conclusions. A stored single-stream figure was wrong by **3.2×**. The best batch throughput available from the incumbent was **330.7 tok/s, not the 343.5 we had on record**: a number that, on inspection, corresponded to no configuration we could run. And the throughput advantage of the candidate moved from a reported **~7× to 17.0×**, or **33.9× per GPU**, since it achieved its figure on one card against the incumbent's two, and had not saturated when our ladder ran out.

The general claim is narrow and, we think, widely applicable: **a recorded performance number is a property of a configuration, not of a model**, and throughput must be measured as a curve against concurrency rather than as a point. We give the diagnostic signature, the method, and the results, including a quality finding that reversed under repetition, and one that did not.

1. A Number That Belonged to No Running System

The work began as a routine evaluation. We had recorded, from an earlier benchmark, that our incumbent on-premises text model (DeepSeek-V4-Flash-0731, served in NVFP4 on two workstation-class Blackwell GPUs) delivered 52.2 tokens per second single-stream and 343.5 tokens per second aggregate at concurrency 16. Against those figures the candidate looked roughly seven times faster in aggregate, and we wrote that down.

Two things about the pair were wrong, and neither was visible without re-measuring.

The live service was returning **165.2 tok/s single-stream**, not 52.2. The stored figure had been captured under a different serving configuration and carried forward as though it described the model. Speculative decoding (enabled on the production unit, absent from whatever run produced the stored number) accounts for essentially all of the difference.

The aggregate figure was worse: it did not correspond to any configuration we could reproduce. The production unit could not reach 343.5 tok/s at any concurrency, and the alternative configuration that can batch peaks at 330.7. The number had become detached from the system it purported to describe, and nothing in our process was positioned to notice, because nobody re-derives a performance figure that already exists.

2. The Signature

Sweeping concurrency rather than sampling it produces a diagnostic that a single point cannot:

Pattern	Interpretation
Aggregate rises with concurrency	The engine is genuinely batching
Aggregate flat beyond N, p95 flat	Saturated at N: N is the real ceiling
Aggregate flat, p95 climbing linearly	Requests are queueing, not running concurrently

The third row is the dangerous one. An engine that queues still reports a respectable aggregate (it is, after all, producing tokens the whole time) and at low concurrency it is indistinguishable from an engine that batches. The distinction only appears when you increase load and watch whether anyone waits.

Our production configuration produced the third pattern exactly. Aggregate throughput was flat within measurement noise across a 64-fold increase in offered load, while per-request latency degraded linearly with it. That is the shape of a queue.

The cause was not a defect. The unit runs `--max-num-seqs 1` because the speculative-decoding path it uses requires it on this stack; raising it triggers a device assertion. The constraint was documented in the unit's own configuration comments. What was missing was any measurement capable of showing what that constraint costs, and any awareness that our recorded batch figure had been captured without it.

The constraint is ours, not the model's. The single-sequence pin is a trade we made, single-stream latency bought at the price of batch capability, and we had been quoting batch figures for a configuration that had already traded batch away. That distinction is this paper's thesis rather than a courtesy: a measurement that describes a configuration cannot be attributed to a model, in either direction.

3. Method

Four serving configurations were measured on an identical ladder: concurrency 1, 2, 4, 8, 16, 32, 64. Each level issues a fixed prompt with a 256-token generation cap, discards a warmup request, then measures wall-clock time across the parallel batch. Aggregate throughput is total completion tokens divided by batch wall time; per-request latency percentiles are retained separately. The client is the same one our quality harness uses, so throughput and quality numbers remain directly comparable.

The configurations:

- **Production incumbent** (DeepSeek-V4-Flash-0731 NVFP4, tensor-parallel across two GPUs, speculative decoding at k=3, `max-num-seqs 1`, 256k context.

Batch-capable incumbent) the same model and weights, no speculative decoding, `max-num-seqs 4`, 64k context. This is the configuration one would enable to get batch throughput from this model.

Candidate, no speculation (Nemotron 3.5 Lightning 30B-A3B NVFP4 on a single GPU, 64k context.

Candidate with drafter) the same, plus NVIDIA's DSpark speculative drafter (a separate 1.35 GB checkpoint) at three speculative tokens.

The candidate is a 30-billion-parameter mixture-of-experts model with roughly 3 billion active parameters, built on a hybrid architecture interleaving Mamba-2 state-space blocks, MoE blocks, and periodic attention. It was served in NVFP4 through the Marlin path: the same quantisation path every NVFP4 baseline on our fleet uses, which keeps the comparison like-for-like on workstation Blackwell, where the vendor's native-FP4 kernels do not apply.

Quality was measured with our internal frontier benchmark: 94 original items across ten dimensions, every verdict produced by code rather than by a language-model judge, with guards that mark truncated responses unscorable rather than failed.

4. Results: Throughput

Aggregate tokens per second:

Conc.	Production incumbent	Batch-capable incumbent	Candidate	Candidate + drafter
1	165.2	110.0	371.0	414.9
2	164.3	187.6	696.5	765.1
4	167.7	328.9	1,077.0	1,317.6
8	166.1	325.6	1,666.2	2,184.9
16	167.1	330.7	2,569.0	2,565.3
32	166.5	320.4	3,718.9	4,335.6
64	165.5	322.9	5,220.4	5,613.4
Peak	167.7 @4	330.7 @16	5,220.4 @64	5,613.4 @64

95th-percentile latency at concurrency 64: **92.73 s**, **47.02 s**, **3.11 s**, **2.86 s** respectively. There were no request errors at any level on any configuration.

Three readings follow directly. The production incumbent does not batch: a flat line across a 64-fold load increase, with latency absorbing all of it. The batch-capable incumbent does batch, and stops at four, exactly its configured limit, after which additional load purchases nothing and costs latency. The candidate had **not saturated at 64**; the ladder ran out before the engine did, so its peak figure is a floor rather than a ceiling.

The headline comparison, 17.0×, is the candidate's peak against the best figure any incumbent configuration produced. At matched concurrency 64 it is 17.4×. At concurrency 1 it is 2.5×. Which number is the honest one depends entirely on the workload, which is the point: **there is no single throughput ratio between two serving configurations, only a curve.**

One figure deserves to be stated on its own terms, because the hardware asymmetry runs against the winner. Nemotron 3.5 Lightning produced 5,613 tokens per second on **one** GPU. The best any incumbent configuration managed was 330.7 across **two**. Normalised per GPU, that is **33.9×**, and it is a floor, because the candidate had not saturated when the ladder ended.

That result is architectural, not incidental. A hybrid design that interleaves Mamba-2 state-space blocks with sparse mixture-of-experts layers and only periodic attention keeps roughly three billion parameters active out of thirty, and the state-space blocks avoid the quadratic attention cost that normally punishes exactly this regime: many concurrent sequences decoding at once. Quantised to NVFP4 the weights occupy 21.58 GB, so the whole model, its speculative drafter, and a 64k-token context fit comfortably on a single workstation-class Blackwell card with room to spare. This is what it looks like when the model architecture, the numerical format, and the silicon are designed against the same target: a 30-billion-parameter model serving sixty-four concurrent users from one GPU, at sub-three-second p95, with no measurable quality penalty.

5. Results: Quality, and a Result That Did Not Survive Repetition

On the 94-item benchmark, the candidate with its drafter scored 84.3 flat, against 84.4 recorded for the incumbent's production configuration: parity, at 2.5× the single-stream speed.

Two quality findings are worth separating, because one is robust and one is not.

Speculative decoding cost the candidate nothing. Running it with and without the drafter produced 84.3 and 81.1. The 3.2-point difference looks like a gain and is not one: only three of ten dimensions moved, by exactly one item each, with the remaining seven identical: three flips across roughly ninety scored items. The finding worth reporting is the absence of a regression, which was the actual risk. Speculative decoding is output-equivalent in principle, but a drafter that overruns a stop token can corrupt output in practice, and we have been bitten by exactly that failure before on a different model. It did not occur here.

The incumbent's batch configuration scored materially below its production configuration, and we cannot yet say why. Because a single run was about to inform a placement decision, we ran it twice: 77.8, then 72.3: both far below the 84.4 on record, with the reasoning dimension collapsing from 100 to 62.5 and then 50.0. Throughput across those same two runs reproduced to within 0.3%. The engine is stable; the answers move.

We are reporting this as measured and unexplained. Two candidate causes were not separated: the batch configuration (no speculation, shorter context, a smaller batched-token budget) may genuinely degrade output, or the model may exhibit run-to-run variance at temperature zero that a single earlier run never surfaced. Either way the operational consequence is the same, and it is the reason the finding is in this paper: **the configuration one would enable to obtain batch throughput from the incumbent scored 72–78, not 84.** A decision made on the recorded figure would have been made on a number that does not describe the system it would have selected.

The asymmetry between the two kinds of measurement is itself instructive. Performance numbers reproduced to a fraction of a percent across repeats; quality numbers moved by five and a half points on the same engine. Repeating a throughput measurement is nearly worthless. Repeating a quality measurement, before it carries a decision, is not optional.

6. Why Speculative Decoding Can Cost You Batching, and Why It Did Not Here

Speculative decoding trades compute for latency: a small drafter proposes several tokens, the full model verifies them in one pass, and accepted tokens are emitted without a separate forward pass each. It is a strong single-stream optimisation, and on our production incumbent it roughly triples single-stream throughput.

It is not free at the scheduler. Draft-token slots consume the same batched-token budget as real sequences, and on some stacks the verification path constrains how many sequences can be in flight. On our incumbent, that constraint is absolute: speculation and batching are mutually exclusive, and enabling one disables the other.

We expected the same trade on the candidate and did not find it. With NVIDIA's DSpark drafter attached it scaled to 5,613 tok/s across 64 concurrent requests while holding p95 under three seconds: faster than the same model without speculation at every level we measured, from one concurrent request to sixty-four. Speculation and batching, mutually exclusive on our incumbent path, compose cleanly here.

That is a harder engineering result than it looks. Speculative decoding degrades as batch size grows (draft tokens compete with real sequences for the same budget, and acceptance rates fall as the scheduler fills) which is why the usual guidance is to disable it above modest concurrency. Delivering a drafter that still pays for itself at sixty-four concurrent sequences, on a hybrid state-space architecture, in a four-bit format, means the drafter, the verification path, the scheduler, and the quantisation were designed against each other rather than bolted together. Vertical integration is easy to claim and hard to measure; this is what it looks like on the instrument.

The transferable lesson is not that speculation is free. It is that **the interaction between speculation and batching is a property of a specific model-and-runtime pairing and must be measured for each**, never inherited from experience with another.

7. Threats to Validity

We state these plainly because several of them bound the headline number.

Asymmetric hardware. The candidate was measured on one GPU, the incumbent on two. The 17× is a comparison of deployable capability on the fleet as it stands, not a per-GPU efficiency claim; per GPU, it understates the candidate.

Decode-bound measurement. Every worker in the sweep receives an identical prompt, so prefix caching makes prompt processing nearly free and the result characterises decode throughput. Mixed production traffic will be lower for all configurations. The ratios are the durable part; the absolute figures are an upper bound.

The ladder is too short. The candidate had not saturated at concurrency 64. We do not know where it does.

Unexplained quality variance. The incumbent's batch-configuration scores rest on two runs with a 5.5-point spread and no established cause. The direction held in both; the magnitude should not be quoted.

Single-run quality elsewhere. Other configurations were scored once, at temperature zero, with no pass-at-k reliability wrapper. Given the variance observed above, this is a weaker guarantee than we would like.

Context. The candidate supports a million-token context and was served at 64k. Long-context behaviour is untested here.

8. What We Changed

Three changes, in increasing order of durability.

We built a concurrency-sweep harness and made it the default way serving configurations are compared, retaining per-request latency percentiles alongside aggregate throughput so that the queueing signature is visible by construction rather than by inference.

We stopped reading configuration state from records. A serving cap is now read off the running process, not from a unit file, a deployment manifest, or an internal memory note: all three were accurate about intent and silent about what was actually loaded.

And we corrected the stored figures, which is the smallest change and the one with the longest reach. A benchmark record answers the question what did this system do on that day, in that configuration. It is routinely read as though it answered what is this model capable of. The two questions diverge the moment anything about the deployment changes, and nothing in a stored number announces which one it is answering.

The uncomfortable form of the lesson: our incumbent had been serially processing every batch workload sent to it, in violation of our own standing engineering rule against exactly that, while a performance figure on record said otherwise. No control was wrong. No alert could have fired. The measurement simply did not have the resolution to contain the failure.

There is a more encouraging form of it too, and it is the reason this paper exists rather than a private ticket. The same sweep that exposed a serial engine also showed a thirty-billion-parameter model serving sixty-four concurrent users from a single workstation GPU, faster with speculative decoding than without it at every point on the curve, at parity on a code-scored quality benchmark. Both facts were invisible to the measurement we had been using. Better instruments do not only find what is broken; they are also the only way to find out that something has become dramatically better than you assumed. We had been reasoning about our own capability from a number captured months earlier, in a configuration that no longer existed, on a stack that had moved underneath it.

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-2026-010. All measurements were taken on production hardware between 14:00 and 15:30 UTC on 11 August 2026; the four sweeps and five benchmark runs described above are retained in full, including raw per-item transcripts, and the production configuration was restored and verified serving after measurement.