PURETENSOR
INTELLIGENCE DISTILLED

Native FP4 Is Slower Than Emulated FP4

A kernel-maturity measurement on workstation Blackwell

Paper: PT-TN-2026-006 v1.0

Author: PureTensor Ltd

Date: 12 March 2026

Status: Published

Abstract

On 11 March 2026, the day NVIDIA released Nemotron 3 Super 120B A12B NVFP4, we served it on a single RTX PRO 6000 Blackwell workstation card and measured throughput under two MoE kernel backends: the native CUTLASS SM120 FP4 path and the older Marlin FP4 weight-only emulation. Native FP4 lost at every concurrency level from 1 to 64. The gap was largest at single-request (83 tok/s native versus 97 tok/s Marlin, a 14% deficit) and smallest at concurrency 16 and 64 (7%), but the sign never flipped. We attribute the slowdown to immature kernel autotuning: during JIT compilation, many CUTLASS tile-shape tactics fail with internal errors, leaving only suboptimal fallbacks.

The measurement also exposed a configuration artefact in our first baseline. An earlier run with --enforce-eager (a defensive flag to avoid CUDA-graph capture crashes) reported 27 tok/s single-request. Disabling that flag and reducing GPU memory utilisation from 0.9 to 0.85 to fit graph buffers yielded 97 tok/s, a 3.6x speedup. The 27 tok/s figure would have been published as the model's speed had we not re-run the comparison. Any conclusion drawn from a day-one number on a day-one stack should be treated as provisional.

Two of the three fastest configuration wins came from disabling the newest code path. The FP8 dense kernel fell back from FlashInfer to CUTLASS; the FP4 MoE kernel fell back from native CUTLASS to Marlin. The priority chains in the serving stack are ordered by intended sophistication, not by measured speed on the installed hardware. Silicon capability is not performance; performance arrives with the kernel autotuner.

Scope is narrow: one GPU, one serving framework at one pinned commit, one FlashInfer version, one model. SGLang 0.5.6 could not serve this model on SM120 at all, so there is no cross-framework control. TP=2 and long-context runs (64k, 128k, 256k) were never measured. We also ran quality and tool-use evaluations against Claude models; those scores rest on small samples and a language-model judge, and the tool-use failure rate may reflect quantisation artefacts rather than architectural limits. The recommendation we recorded at the time: watch FlashInfer releases for SM120 autotuner improvements and re-run the sweep.

1. Hardware, model, and software stack

The card is an RTX PRO 6000 Blackwell, 96 GB, compute capability 12.0 (SM120). A second identical card sat idle, reserved for a possible TP=2 test that was never run. The model, NVIDIA

Nemotron 3 Super 120B A12B NVFP4, is a 120B-parameter mixture-of-experts architecture with 12B active parameters and 512 routed experts (22 active per token). Its hybrid design combines Mamba-2 state-space layers, MoE blocks, and attention. Weights occupy 75 GB on disk across 17 safetensor shards. Maximum context is 262,144 tokens. NVFP4 means FP4 weights with FP8 activations; the KV cache is FP8.

The software stack was pinned by the model vendor: NVIDIA driver 580.126.20, CUDA 12.8, PyTorch 2.10.0+cu128, vLLM 0.16.1rc1.dev206+g097eb544e, FlashInfer 0.6.6, Python 3.12, Linux 6.17.0-14-generic. SGLang 0.5.6 lacks SM120 kernel support and was not a usable alternative. The vendor mandates temperature 1.0 and top_p 0.95 for sampling.

Memory headroom matters. With --gpu-memory-utilization=0.85, vLLM allocates roughly 81 GB for weights and KV cache, leaving about 15 GB for CUDA-graph capture and PyTorch workspace. At 0.9, graph capture runs out of memory. The 0.85 threshold is the proven safe value on this card.

Serving flags of record: --async-scheduling, --kv-cache-dtype fp8, --attention-backend TRITON_ATTN (FlashInfer attention has SM120 compatibility issues), --enable-chunked-prefill, --trust-remote-code, --max-num-seqs 256, and the environment variable VLLM_SLEEP_WHEN_IDLE=1.

2. Launch failures and their workarounds

Two errors blocked the first launch.

The FlashInfer FP8 batch-matmul kernel failed during kernel_warmup() in the vLLM GPU worker. The message was terse: bmm_fp8_internal_cublaslt failed: the library was not initialized. The cause: FlashInfer's bmm_fp8() calls an SM100 (Hopper) code path, and the cuBLASLt FP8 batch-matmul kernel does not exist for SM120 in FlashInfer 0.6.6. The workaround is to set VLLM_DISABLED_KERNELS=FlashInferFP8ScaledMMLinearKernel, forcing the CutlassFP8 dense-linear backend.

The FlashInfer MoE FP4 CUTLASS autotuner failed during JIT. The logs are full of lines like flashinfer.jit: [Autotuner]: Skipping tactic … due to Error Internal: cutlass_kernel_file_gemm_grouped_sm120 …. Many tile-shape tactics fail, leaving only suboptimal fallbacks. The workaround is VLLM_USE_FLASHINFER_MOE_FP4=0, which falls back to the Marlin MoE backend (FP4 weight-only emulation). Under Marlin the serving stack prints a cosmetic warning: "Your GPU does not have native support for FP4". The hardware does have native FP4; the warning fires because the environment flag forces the emulated path.

Several smaller issues also appeared. A flashinfer-cubin version mismatch (0.5.3 installed against flashinfer-python 0.6.6) required installing flashinfer-cubin 0.6.6 and setting FLASHINFER_DISABLE_VERSION_CHECK=1. Graph capture crashed at utilisation 0.9; the fix was to reduce to 0.85. The engine crashed under concurrent evaluation load at --max-num-seqs 512; reducing to 256 and staggering evaluation clients fixed it.

The kernel-selection chains vLLM traverses on SM120 are instructive. For dense FP8: FlashInferFP8ScaledMMLinearKernel (broken), CutlassFP8ScaledMMLinearKernel (working, used), PerTensorTorchFP8, ChannelWiseTorchFP8. For MoE FP4: FLASHINFER_TRTLLM (autotuner failures), FLASHINFER_CUTEDSL (autotuner failures), FLASHINFER_CUTLASS (partial, 10–15% slower than Marlin), VLLM_CUTLASS (available), MARLIN (working, fastest currently). The chains are ordered by intended sophistication, not by measured speed.

3. The enforce-eager baseline (why the first number was wrong)

The initial configuration carried --enforce-eager, which disables CUDA-graph capture. That flag was added defensively to avoid graph-capture crashes at higher memory utilisation. The published baseline from that run: 27 tok/s single-request, 736 tok/s aggregate at 32 concurrent.

Removing --enforce-eager and dropping utilisation from 0.9 to 0.85 to fit graph buffers gave 97 tok/s single-request and 886 tok/s at 32 concurrent, a 3.6x speedup at low concurrency. Kernel-launch overhead per forward pass is large for an MoE model with many small expert kernels; CUDA graphs amortise that overhead.

Configuration	Single tok/s	Aggregate at 32 concurrent
--enforce-eager, 0.85–0.9 util (no CUDA graphs)	27	736
No --enforce-eager, 0.85 util (CUDA graphs)	97	886

The enforce-eager run had more detail: single-request throughput was 26.2 tok/s on short prompts, 27.5 on medium, 27.2 on long. Mean time to first token was 6.20 s for short prompts, 6.78 s for medium. GPU memory sat at 81 GB of 96 GB. The concurrency curve was 27.1 / 53.6 / 111.3 / 178.6 / 395.5 / 736.2 tok/s at 1 / 2 / 4 / 8 / 16 / 32 concurrent requests, with P50 latency 9.5 / 9.6 / 9.2 / 11.5 / 10.4 / 11.1 s. Scaling was near-linear to 16x; at 32x concurrency, throughput was 27x the single-request rate with an 18% latency increase.

The caveat is plain: the 27 tok/s figure was a configuration artefact, not a hardware ceiling. Had we not re-run the comparison, it would have been reported as the model's speed.

4. Native CUTLASS SM120 FP4 versus Marlin emulation

With CUDA graphs enabled, we swept concurrency from 1 to 64 on both backends.

Concurrency	Marlin FP4 (tok/s)	Native CUTLASS FP4 (tok/s)	Delta
1	97	83	−14%
2	147	131	−11%
4	246	225	−9%
8	375	342	−9%
16	574	532	−7%
32	886	801	−10%
64	865	803	−7%

Native CUTLASS SM120 FP4 kernels are 10–15% slower than Marlin FP4 weight-only emulation across the board. The deficit is largest at concurrency 1 (14%) and smallest at 16 and 64 (7%), but the sign never flips. Throughput peaks at 32 concurrent on both backends (886 Marlin, 801 native) and falls slightly or stays flat at 64 (865 Marlin, 803 native, within noise).

The cause is immature kernel autotuning. Many CUTLASS tile-shape tactics fail with Error Internal during JIT compilation, leaving only suboptimal fallbacks. The decision we recorded: revert to the Marlin FP4 backend as the faster option; re-test when FlashInfer ships improved SM120 autotuning. Native FP4 is expected to eventually outperform Marlin once the autotuner matures, but that expectation is a hypothesis, not a measurement.

5. Quality and tool-use evaluations

We ran quality and tool-use evaluations against the same served model, with Claude Opus 4.6 and Claude Sonnet 4.6 as comparisons. The numbers are included here because they were collected in the same session, but the sample sizes are small and the judge is a language model.

Quality evaluation used 20 prompts scored 1–10 by a Claude judge. On reasoning (10 prompts), the 120B MoE scored 9.58 versus 9.88 for Opus and 9.76 for Sonnet. On coding (fewer prompts, exact count not recorded), it scored 8.32 versus 7.48 for Opus and 8.10 for Sonnet. Average response times were longer: reasoning 37.2 s versus 14.0 s (Opus) and 9.7 s (Sonnet); coding 82.5 s versus 68.0 s and 37.7 s. The MoE model generated more tokens on reasoning (1,006 versus 536 and 553) and fewer on coding (2,373 versus 3,106 and 3,196).

Tool-use evaluation used 30 tests across three tiers. On single-tool tests (10), the MoE scored 50% versus 80% for Sonnet. On sequential multi-tool tests (10), 30% versus 70%. On complex agentic tests (10), 30% versus 40%. Overall: 37% versus 63%. The tier-1 JSON error rate was 40% for the MoE versus 0% for Sonnet. Average response times by tier were 25.6 / 39.7 / 27.7 s for the MoE versus 7.0 / 12.7 / 14.3 s for Sonnet. We observed over-calling (5+ tool calls where 1–2 would suffice) and intermittent "Server disconnected" errors during long multi-tool chains.

The 40% tier-1 JSON error rate was attributed in our notes to a quantisation artefact. That attribution is a hypothesis, not a measurement. The coding win rests on the fewest samples. The quality scores come from a language-model judge with n=10 for reasoning and fewer for coding. These numbers are directional at best.

6. What survives and what does not

The native-versus-Marlin comparison survives scrutiny. It is a full concurrency sweep, both backends, same model, same card, same session, same software stack. The 10–15% deficit is consistent in sign across seven concurrency levels. The mechanism (autotuner failures leaving suboptimal fallbacks) is visible in the logs.

The 3.6x CUDA-graph speedup survives as a measurement but carries a warning: it is the difference between a misconfigured run and a correctly configured one, not a comparison of two valid configurations. The 27 tok/s figure should not be cited as the model's speed.

The quality and tool-use evaluations do not survive as strong claims. Sample sizes are small (10 reasoning prompts, 10 tests per tool-use tier, fewer coding prompts). The judge is a language model. The JSON error rate hypothesis (quantisation artefact) is untested. These numbers are included for completeness; they are not the point of the note.

Scope is narrow throughout. One GPU. One serving framework at one pinned commit. One FlashInfer version. One model. SGLang 0.5.6 could not serve this model on SM120, so there is no cross-framework control. TP=2 was never run. Long-context runs (64k, 128k, 256k) were never run. The recommendation we recorded at the time stands: watch FlashInfer releases for SM120 CUTLASS FP4 autotuner improvements and re-run this exact sweep. The expected outcome is an eventual inversion, but expected outcomes have a way of not arriving.

PureTensor operates a sovereign AI infrastructure fleet (on-premises GPU compute, storage, and serving) run day-to-day by autonomous agents under human direction. PT-TN-2026-006. Measurements taken 11 March 2026, 19:00–22:13 UTC. Raw artefacts are retained.