On One RTX PRO 6000 Blackwell, Compiled NVFP4 Takes 2.78× as Long as FP16 at 1×512 and Is 1.28× Faster Than the Best 16-Bit Path at 32×512

Dmitrii Lozhkin
AVELIN AI

Abstract

Lower precision can reduce memory without accelerating an individual decision. We time RoBERTa-large-MNLI on one NVIDIA RTX PRO 6000 Blackwell GPU and four cores of an Intel Xeon Platinum 8559C CPU. At batch 1 and length 512, compiled NVFP4 W+A takes 9.43 ms versus 3.39 ms for compiled FP16. At 32×512, NVFP4 W+A delivers 1,112 decisions/s, 1.28× the best measured 16-bit path (871). Compiled FP8 W+A reaches 1,100 decisions/s at 32×512 (1.26×), within 1.0% of NVFP4, and is faster than NVFP4 at 1×512 and 8×256. Compilation also changes decisions: on 300 real inputs, INT8, FP8 and NVFP4 W+A change 3, 2 and 8 of 300 decisions relative to the same recipe run eagerly. On the CPU, BF16 has lower warm latency than either dynamic INT8 recipe. Speed, memory and decision quality must be checked on the intended execution path.

1 Introduction

FP8 [1] and NVFP4 [2] formats and quantized transformer methods [3] offer more compact arithmetic, but performance depends on scale computation, kernels and workload shape. An evaluation must distinguish latency from throughput and check individual decisions, because accuracy can conceal opposing transitions. We measure both on one public model and keep the decisions produced by every timed path.

2 Setup

We use RoBERTa-large-MNLI revision `2a8f12d2` [4,5] and MultiNLI development data revision `da70db2a` [6,7]. All runs use one cloud instance with an NVIDIA RTX PRO 6000 Blackwell Server Edition (96 GB, sm_120), driver 595.112.07, and an Intel Xeon Platinum 8559C with 16 logical / 8 physical cores and 128 GiB RAM. Software is Python 3.12.3, PyTorch 2.14.1+cu130, CUDA 13.0, torchao 0.18.0, Transformers 5.21.0 and NumPy 2.5.4. TF32 is disabled and the final classifier stays FP32.

GPU low-bit recipes first convert the encoder to BF16, then quantize its Linear layers with torchao 0.18.0 [8]: INT8 W+A uses `Int8DynamicActivationInt8WeightConfig()`, FP8 W+A uses `Float8DynamicActivationFloat8WeightConfig(granularity=PerRow())` with fast accumulation, and NVFP4 W+A uses `NVFP4DynamicActivationNVFP4WeightConfig(use_triton_kernel=False, use_dynamic_per_tensor_scale=True)`. Compiled paths use `torch.compile` [9] in the default Inductor mode, without CUDA graphs, with `fullgraph=True` and `dynamic=False`. Every input shape is compiled and warmed before timing; a timed region with any compilation or recompilation would be discarded, and none occurred. CPU tests use four pinned physical cores; CPU INT8 uses FBGEMM through `quantize_dynamic` with the module's default `reduce_range=True`, with per-tensor (default) or per-channel weights.

Prepared inputs of shape B×L (batch × length in tokens) are constructed from a stream of real MultiNLI tokens, with every position attended; their timer covers the model call, argmax and transfer of decisions to the host. Each shape receives 20 warmup and 100 timed calls. **Real inputs** are 300 development rows, 150 per split, drawn with seed 20261027 and timed one at a time; their timer adds tokenization, padding and transfer to the device. They are padded to execution buckets of 64, 256 or 512 tokens (bucketed padding); 267 fall in bucket 64 and 33 in bucket 256. Decisions/s is completed decisions divided by summed timed seconds. Each configuration runs in three fresh processes; we report means of the process summaries. Process minimum–maximum ranges are within 7% of the mean for every value reported here and within about 1% for 32×512 throughput.

Definitions

A **decision** is the argmax over the three class scores. A **change** is a different decision from the stated reference: in Table 2 the same recipe run eagerly with the same bucketed padding, in Table 3 CPU FP32 on the same inputs. **Corrected**, **newly incorrect** and **wrong-to-different-wrong** mean wrong→right, right→wrong and wrong→another wrong label. **W+A** means quantized weights and activations. **Eager** executes operations one by one; **compiled** uses torch.compile.

3 Results

The winning path changes across measured operating points. Compiled NVFP4 W+A takes 2.78× as long as compiled FP16 at 1×512. At 32×512 it is 1.36× faster than compiled FP16 but only 1.28× faster than eager BF16, the best tested 16-bit path. Compiled FP8 W+A is within 1.0% of NVFP4 at 32×512 (1,100.4 versus 1,111.9 decisions/s; the process ranges do not overlap), 2.0× faster at 1×512 (4.71 versus 9.43 ms) and 1.8× faster at 8×256. At 8×256, FP8 exceeds eager FP16 throughput by 9.42%; NVFP4 remains slower. Compiled NVFP4 latency stays between 9.26 and 9.58 ms from a single 64-token input to 8×256 and on real inputs, which suggests a fixed per-call cost. These are operating points, not a crossover threshold.

Table 1: Prepared GPU inputs; mean process p50 latency (ms) and throughput (decisions/s) across three processes. Latency is for the complete batch. E = eager; C = compiled. INT8, FP8 and NVFP4 use W+A recipes.

Recipe / path	1×512 p50	8×256 p50	8×256 decisions/s	32×512 decisions/s
FP32 / E	11.27	28.00	285.7	165.2
FP32 / C	11.65	28.26	283.0	165.1
FP16 / E	5.12	5.73	1,392.7	856.8
BF16 / E	5.18	5.81	1,379.4	871.4
FP16 / C	3.39	6.21	1,294.7	815.3
BF16 / C	3.43	6.23	1,283.5	828.7
INT8 / C	4.80	6.18	1,291.9	761.5
FP8 / C	4.71	5.24	1,523.9	1,100.4
NVFP4 / C	9.43	9.51	840.6	1,111.9

Compilation is neither uniformly faster nor decision-preserving. At 32×512 both 16-bit eager paths outperform their compiled counterparts. FP32 improves with compilation only for short inputs: from 5.99 to 4.30 ms at 1×64 and from 7.12 to 4.56 ms on real inputs. Table 2 pairs latency with the decisions of the timed path; the three processes produce identical decisions on the same 300 rows.

Table 2: Warm GPU real-input batch-1 p50, mean of three process medians. Correct counts describe the timed path. "Changes" compares compiled with eager decisions of the same recipe, both with bucketed padding, on these 300 rows.

Recipe	Eager ms	Compiled ms	Correct / 300: E → C	Changes
FP32	7.12	4.56	269 → 269	0
FP16	6.31	3.66	269 → 269	0
BF16	6.32	3.67	269 → 269	0
INT8 W+A	166.01	5.03	269 → 270	3
FP8 W+A	38.66	5.92	270 → 272	2
NVFP4 W+A	82.80	9.58	271 → 272	8

The 8 NVFP4 changes comprise 3 newly incorrect, 4 corrected and 1 wrong-to-different-wrong. An exploratory follow-up on a separate balanced 1,200-row set (200 rows per split and label, seed 1729; 18 rows shared with the 300) finds 12 changes for INT8, 6 for FP8, 27 for NVFP4 and 2 for BF16, and none for FP32 and FP16.

Eager low-bit latency includes substantial execution overhead. Separate 1×64 profiles record 6,913 cudaStreamSynchronize and 5,185 aten::item calls per INT8 forward in eager mode, versus none when compiled. This does not isolate the cause of the timing gap.

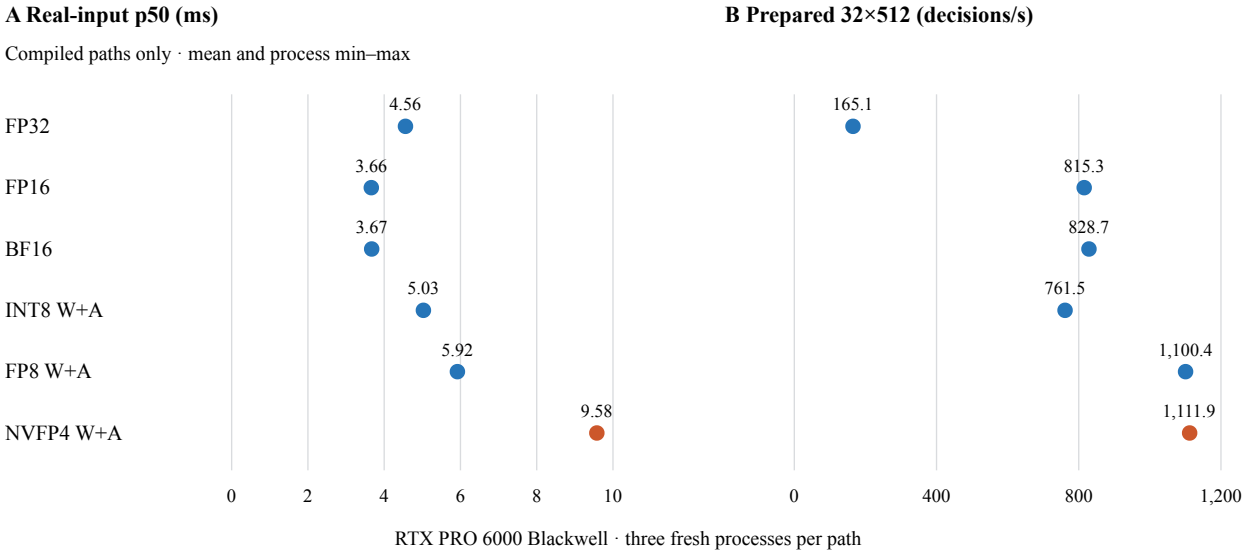


Figure 1: Compiled GPU paths. Left: real-input batch-1 p50 on 300 rows; right: prepared 32×512 throughput. Points show process means; the minimum–maximum ranges are too small to be visible at this scale. Different inputs and timer scopes prevent attributing the contrast to batch size alone.

CPU BF16 wins this latency comparison. The host CPU reports the amx_bf16, amx_int8 and avx512_vnni flags [10]; we did not profile which oneDNN kernels ran.

Table 3: CPU eager results on four cores, batch 1. Latency and timed correctness use the 300 real rows with bucketed padding. A separate 1,200-row set (599 matched and 601 mismatched rows, seed 20261027, containing all 300) uses padding to a multiple of 16 only; its changes are versus CPU FP32 on the same inputs.

Recipe	p50 ms	Correct / 300	Changes / 1,200	Correct / 1,200
FP32	101.91	269	Reference	1,092
BF16	35.87	270	0	1,092
Per-tensor INT8 (quantize_dynamic default)	41.12	268	32	1,089
Per-channel INT8	41.23	269	29	1,090

BF16 agrees with CPU FP32 on all 1,200 rows, yet has one more correct answer on the 300 timed rows: bucketed padding changes one BF16 decision from wrong to right, while the 1,200-row set does not use bucketed padding. Both INT8 recipes keep reduce_range=True; a separate CPU experiment shows that this flag alone accounts for most of their disagreement with FP32.

Allocated memory falls while reservation need not. After model loading in eager processes, allocated GPU memory is 1,355.6 MiB for FP32, 680.6 for FP16 and 266.7 for NVFP4 W+A; reserved memory is 1,368.0 for FP32 versus 1,546.0 for NVFP4 W+A.

4 Limitations

One checkpoint, task and host cannot establish universal rankings. MultiNLI development splits are not an untouched test set. Three process repeats are not independent hardware trials. The real cohort has median length 38 tokens, 89% in bucket 64 and none in bucket 512. Eager low-bit paths include unfused operations. Warm timings exclude loading, compilation, service queues, network transport and concurrency. A different driver, library or device is a new condition.

5 Recommendations

1. Benchmark latency and throughput at the intended shape.
2. Compare against the best measured higher-precision path.
3. Check decisions on the actual timed path; report memory separately.

Disclosure.
Experiments and analysis were produced with AI coding agents under the author's direction. All numbers were recomputed from the saved timings and per-example predictions. The model card specifies the MIT license; MultiNLI combines sources with different licenses, listed in its card [5,7].

References

1. P. Micikevicius, D. Stosic, N. Burgess, et al. FP8 formats for deep learning. arXiv:2209.05433, 2022.
2. NVIDIA, F. Abecassis, A. Agrusa, et al. Pretraining large language models with NVFP4. arXiv:2509.25149, 2025.
3. O. Zafrir, G. Boudoukh, P. Izsak, M. Wasserblat. Q8BERT: quantized 8-bit BERT. arXiv:1910.06188, 2019.
4. Y. Liu, M. Ott, N. Goyal, et al. RoBERTa: a robustly optimized BERT pretraining approach. arXiv:1907.11692, 2019.
5. FacebookAI. FacebookAI/roberta-large-mnli. Hugging Face model card, revision 2a8f12d27941090092df78e4ba6f0928eb5eac98. Accessed October 28, 2026.
6. A. Williams, N. Nangia, S. R. Bowman. A broad-coverage challenge corpus for sentence understanding through inference. NAACL-HLT 2018, pp. 1112–1122. DOI: 10.18653/v1/N18-1101; arXiv:1704.05426, 2018.
7. NYU Machine Learning for Language. nyu-mll/multi_nli. Hugging Face dataset card, revision da70db2af9d09693783c3320c4249840212ee221. Accessed October 28, 2026.
8. PyTorch contributors. TorchAO 0.18.0 (released August 3, 2026): quantization APIs and source code. Git revision 5f2baf9d575cf732362594c998c399902942531f.
9. J. Ansel, E. Yang, H. He, et al. PyTorch 2: faster machine learning through dynamic Python bytecode transformation and graph compilation. ASPLOS 2024. DOI: 10.1145/3620665.3640366.
10. Intel Corporation. Accelerate AI with Intel Advanced Matrix Extensions. Intel quick-start guide, document 794441. Accessed October 28, 2026.