One INT8 Activation-Range Flag Cuts Disagreement With FP32 From 17 to 1 of 600 Batch-1 RoBERTa-large-MNLI Decisions on a VNNI CPU, Without Improving Accuracy

Dmitrii Lozhkin
AVELIN AI

Abstract

Bit width alone does not specify a quantization experiment: activation ranges, scale placement, rounding and surrounding arithmetic can change individual decisions while leaving aggregate accuracy nearly unchanged. On 600 MultiNLI development inputs, switching one activation-range flag on unchanged packed per-channel INT8 weights of RoBERTa-large-MNLI reduces disagreement with CPU FP32 from 17 to 1 at batch 1, while correct answers fall from 535 to 534. At batch 32, disagreement falls from 21 to 3 and correctness from 539 to 536. The flag changes agreement with FP32, not accuracy, and on x86 CPUs without VNNI instructions the full range risks 16-bit overflow. Among GPU recipes, equal-looking change counts hide different changed inputs. We separate reference fidelity from correctness and report complete recipes, paired transitions and input-level overlap on identically aligned inputs.

1 Introduction

Microscaling formats specify elements and scales [1,2]; NVFP4 adds a tensor-level scale [3]. Our earlier note separated scale precision and block size as contributors to weight reconstruction error [4]; those measurements do not establish their effects on model decisions. Dutta et al. showed that compressed language models with near-baseline accuracy flip many answers between correct and incorrect and proposed reporting such flips [5]. Following [5], we separate agreement with a reference from labeled correctness and test which recipe details, beyond bit width, change individual decisions.

2 Setup

We use RoBERTa-large-MNLI revision `2a8f12d2` [6] and MultiNLI development data revision `da70db2a` [7]. All runs use one cloud instance with an Intel Xeon Platinum 8559C, whose reported flags include `avx512_vnni`, and an NVIDIA RTX PRO 6000 Blackwell Server Edition (96 GB, sm_120), driver 595.112.07. 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; the classifier head stays FP32.

The CPU experiment uses 600 development rows: 100 per split and label, chosen by a seeded hash (seed 1729) and kept in dataset order. They are a subset of a balanced 1,200-row set used for batch checks and share 8 rows with a 300-row latency cohort and 37 with a 1,200-row CPU quality set in our timing study. Inputs are truncated at 512 tokens; at batch 32, consecutive rows form a batch, right-padded to its longest row and rounded up to a multiple of 16. CPU INT8 uses FBGEMM on four pinned physical cores and converts the encoder's Linear layers with `quantize_dynamic`, using per-tensor (default) or per-channel weights. GPU comparisons use all 19,647 rows in eager batches of 32, with low-bit Linear layers inside a BF16 encoder.

Definitions

A **decision** is the argmax over the three class scores; a **change** is a different decision from the stated FP32 reference. **Corrected**, **newly incorrect** and **wrong-to-different-wrong** mean wrong→right, right→wrong and wrong→another wrong label. **W+A** denotes quantized weights and activations. **Jaccard** is the size of the intersection of two changed-input sets divided by the size of their union. The **gap** is the difference between the two largest FP32 class probabilities.

3 Results

Closer agreement need not improve accuracy. The dynamic Linear module of the installed PyTorch (module version 4, the current one) passes reduce_range=True to its operator [8]. We call the same operator on the same packed weights with reduce_range=False, which changes the nominal unsigned activation bounds from 0–127 to 0–255 [9]. Both runs use the same packed objects: one SHA-256 digest over the quantized weights, scales, zero-points, axes and biases of all 144 dynamic Linear modules is identical before and after the change.

Table 1: CPU recipe experiment on the same 600 labeled inputs. Changes use CPU FP32 at the same batch size; both per-channel rows share packed weights. Cells report batch 1 / batch 32 counts. Reduced and full refer to activation range.

Recipe	Changed / 600	Correct / 600
CPU FP32	Reference	535 / 535
Per-tensor INT8 (quantize_dynamic default), reduced	18 / 25	536 / 540
Per-channel INT8, reduced	17 / 21	535 / 539
Per-channel INT8, full	1 / 3	534 / 536

Against reduced range, full range changes 16 decisions at batch 1: 8 newly incorrect, 7 corrected and 1 wrong-to-different-wrong. At batch 32 the corresponding counts are 22, 12, 9 and 1. These accuracy differences are within noise (exact McNemar p = 1.0 and 0.66). Full range was tested only with per-channel weights.

The reduced range protects CPUs without VNNI. PyTorch reduces the range so that the x86 vpmaddubsw instruction does not overflow [10]: it sums pairs of 8-bit products into a saturating 16-bit integer, and $255 \cdot 127 \cdot 2 = 64{,}770$ exceeds 32,767. Our CPU has VNNI, whose instructions accumulate into 32 bits, so the full-range result applies only to such CPUs; on older x86 CPUs full range can saturate.

Whole-tensor ranges couple decisions within a batch. FBGEMM computes one activation min/max over the whole input tensor, including padded positions [11]. Batch 1 versus batch 32 on the same 600 rows changes no CPU FP32 decisions, but 19 per-tensor INT8, 23 reduced-range per-channel and 4 full-range per-channel decisions. The tested GPU NVFP4 W+A recipe likewise computes its activation tensor scale over the whole batch [12].

Matching totals is weaker than matching changed inputs. Table 2 contrasts whole recipes against the same FP32 run: weights-only with W+A within INT8 and NVFP4, and FP8 with INT8 W+A.

Table 2: Changed-set intersections and Jaccard overlaps for GPU recipes on 19,647 inputs, eager batch 32. W = weights only; W+A = weights and activations. All recipes use BF16 surrounding arithmetic.

Pair	Changed totals	Intersection / union	Jaccard
INT8 W / INT8 W+A	74 / 291	38 / 327	11.62%
FP8 W+A / INT8 W+A	147 / 291	55 / 383	14.36%
NVFP4 W / NVFP4 W+A	501 / 738	378 / 861	43.90%

Four-bit recipes differ beyond their elements. MXFP4 uses E2M1 elements, blocks of 32 and E8M0 scales [1]; NVFP4 uses blocks of 16, E4M3 local scales and an FP32 tensor scale [3]. In the tested NVFP4 source path, the two FP32 global scales are multiplied, their product is cast to BF16 and applied after the matrix multiplication, before the bias [12].

Small gaps concentrate changes but do not contain most of them. All 3 FP16 changes have a gap below 0.01. NVFP4 W+A changes 346 of 801 decisions with a gap below 0.25 and 392 of 18,846 at or above it: 43.20% versus 2.08%. Most NVFP4 changes (392 of 738, 53%) therefore lie at gaps of at least 0.25, where a filter flagging only gaps below 0.25 would miss them. The gap is not calibrated confidence.

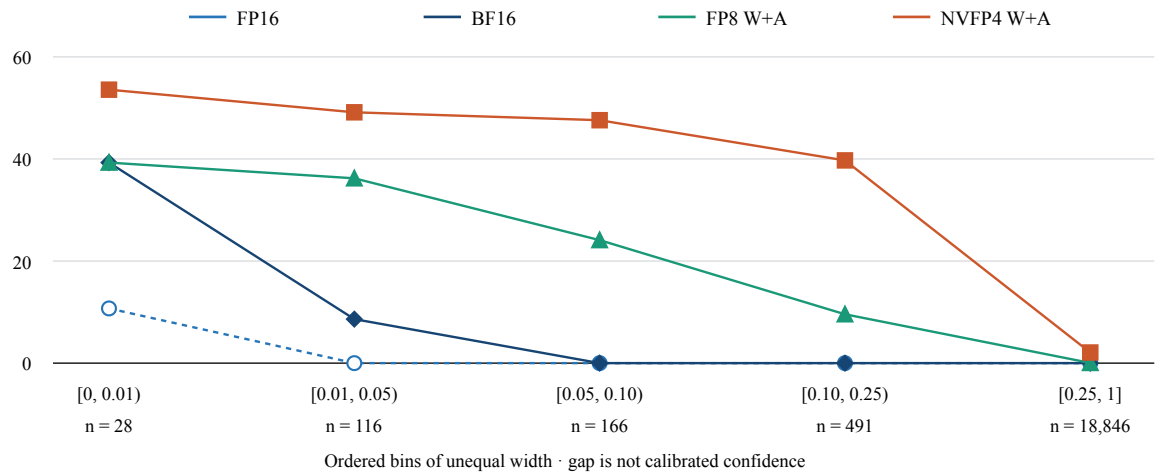


Figure 1: Eager batch-32 GPU changes by FP32 gap, percent of each bin; bins have unequal widths, the last is closed at 1.

4 Limitations

One checkpoint and development task limit generalization. Batching also changes padding and possible dispatch. The batch-32 GPU run was not repeated; batch-1 reruns of six GPU recipes on a 1,200-row subset reproduced the saved logits bitwise, so we expect, but have not verified, identical changed sets on a rerun. The weight digest covers the logical quantized parameters, not a fresh export of the packed bytes. No universal ranking or downstream utility follows.

5 Recommendations

1. Record scales, ranges, rounding and surrounding arithmetic alongside bit width.
2. Separate reference fidelity, labeled accuracy and changed-set overlap.
3. Recheck batch and padding changes on paired inputs; keep reduce_range=True on CPUs without VNNI.

Disclosure.
Experiments and analysis were produced with AI coding agents under the author's direction. All numbers were recomputed from the saved per-example predictions. The model card specifies the MIT license; MultiNLI combines sources with different licenses, listed in its card [6,7].

References

1. Open Compute Project. OCP microscaling formats (MX) specification, version 1.0. September 2023.
2. B. D. Rouhani, R. Zhao, A. More, et al. Microscaling data formats for deep learning. arXiv:2310.10537, 2023.
3. NVIDIA, F. Abecassis, A. Agrusa, et al. Pretraining large language models with NVFP4. arXiv:2509.25149, 2025.
4. Y. Akinin. NVFP4 versus MXFP4 on synthetic weights: the scale's precision alone closes two thirds of the error gap, the smaller block alone a quarter at most. AVELIN AI Lab Research Note, October 2026.
5. A. Dutta, S. Krishnan, N. Kwatra, R. Ramjee. Accuracy is not all you need. NeurIPS 2024; arXiv:2407.09141, 2024.
6. FacebookAI. FacebookAI/roberta-large-mnli. Hugging Face model card, revision 2a8f12d27941090092df78e4ba6f0928eb5eac98. Accessed October 28, 2026.
7. NYU Machine Learning for Language. nyu-mll/multi_nli. Hugging Face dataset card, revision da70db2af9d09693783c3320c4249840212ee221. Accessed October 28, 2026.
8. PyTorch contributors. Dynamic quantized Linear module (torch/ao/nn/quantized/dynamic/modules/linear.py). PyTorch source revision 5c4886908584029761b579af026dcfb627c84070, 2026.
9. PyTorch contributors. CPU quantization utilities (aten/src/ATen/native/quantized/cpu/QuantUtils.h). PyTorch source revision 5c4886908584029761b579af026dcfb627c84070, 2026.
10. PyTorch contributors. Quantization observers (torch/ao/quantization/observer.py). PyTorch source revision 5c4886908584029761b579af026dcfb627c84070, 2026.
11. PyTorch contributors. FBGEMM dynamic Linear implementation (aten/src/ATen/native/quantized/cpu/qlinear_dynamic.cpp). PyTorch source revision 5c4886908584029761b579af026dcfb627c84070, 2026.
12. PyTorch contributors. NVFP4 tensor implementation (torchao/prototype/mx_formats/nvfp4_tensor.py). TorchAO 0.18.0, git revision 5f2baf9d575cf732362594c998c399902942531f, 2026.