NYOOMFLOAT12: LOSSLESS 12-BIT WEIGHT COMPRESSION FOR POST-TRAINING INFERENCE

Anonymous authors
Paper under double-blind review

ABSTRACT

Scaling reinforcement learning post-training requires generating rollouts efficiently, but GPU memory constrains both the speed and scale of this generation. We present NyoomFloat12 (NF12), a lossless 12-bit fixed-length format for BF16 weights that addresses two GPU memory constraints simultaneously. By analyzing per-bit entropy in trained weight exponents, we find that the upper exponent nibble is nearly constant (`0111`); dropping it yields a fixed-length format with 13-operation SIMT-friendly decode. In the memory-bound regime, fusing decode into single-sequence inference achieves up to 1.11× end-to-end speedup (1.28× per GEMV layer). In the memory-constrained regime, the 25% weight footprint reduction frees capacity for larger batch sizes: on Qwen3-32B with SGLang (Zheng et al., 2023), buffered NF12 reaches 1.67× higher rollout throughput as compression allows batch size to scale. NF12 is lossless: escape groups are zeroed during bulk decode and overwritten with the stored originals, producing bitwise-exact BF16 output.

1 INTRODUCTION

Reinforcement learning post-training has become a major driver of LLM capability (DeepSeek-AI, 2025), but scaling it can be bottlenecked by rollout generation. Each RL training step generates thousands of tokens from the policy model, and generating each token requires loading the full weight matrix from GPU memory. On modern GPUs, this memory access, not arithmetic, often determines token generation speed (Recasens et al., 2025).

GPU memory can constrain rollout generation in two ways. At low batch sizes, each token generation scans the full weight matrix from HBM, making GEMV memory-bandwidth bound; the GPU's arithmetic units idle while waiting for data. At high batch sizes, total GPU memory limits how many rollout sequences can run concurrently: weights, KV caches, and activations share a fixed pool (Kwon et al., 2023). Lossy quantization (FP8, INT8) addresses this by reducing precision, but in RL post-training the precision mismatch between quantized rollout weights and full-precision training weights can cause training instability and non-convergence (Xi et al., 2026). Lossless compression avoids this: it reduces memory pressure without changing the weights, if decompression is fast enough.

Making decompression fast enough is a key technical challenge. GPUs execute instructions across 32-thread *warps* in lockstep, so the format must be fixed-length: variable-length codes serialize execution, since each thread processes a different number of bits. Decode should use only register-level arithmetic; lookup tables that accelerate Huffman decoding on CPUs do not help on GPU architectures optimized for arithmetic throughput, not random memory access. Furthermore, the solution should address both memory bottlenecks (bandwidth and capacity) since RL workloads shift between regimes as batch size varies.

Prior lossless methods miss a structural property of BF16 exponents: entropy is not uniform across bits. The upper nibble is nearly constant at `0111`, while the lower nibble retains full entropy (Figure 1). DFloat11 (Zhang et al., 2025) and ECF8 (Yang et al., 2025) encode the exponent as a single 8-bit field, obscuring this per-bit structure; Huffman coding achieves $\sim$11 bits (31% compression) but produces variable-length codes that cannot be fused into compute kernels, forcing an HBM round-trip that offsets bandwidth savings. The per-bit structure enables a fixed-length format that eliminates variable-length decoding entirely.

We present NyoomFloat12 (NF12), a lossless 12-bit fixed-length format for BF16 weights. NF12 drops the constant upper exponent nibble and stores the remaining 12 bits unchanged. Decode requires only 13 SIMT-friendly operations for 8 weights: no lookup tables, no branches, no warp divergence (§2.1). The $<$0.5%

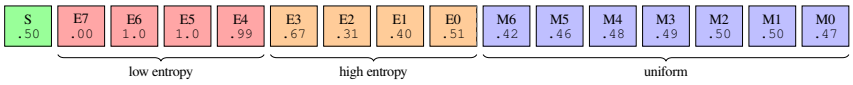

Figure 1: $P(\text{bit}=1)$ for BF16 weights, averaged across 9 models (0.5B–14B). The upper exponent nibble (E7–E4) is almost always `0111` ($H < 0.02$ bits each). NF12 drops this nibble.

of weights with out-of-range exponents are stored exactly and overwrite bulk-decoded values, producing bitwise-identical output (§2.1). NF12 operates in two deployment modes: *fused* decode directly to registers during GEMV for the bandwidth-bound regime (§2.2), and *buffered* decode to a reusable HBM buffer for the capacity-bound regime, where the 25% footprint reduction frees memory for larger batches (§2.2).

On H100: (1) buffered NF12 reaches 1.67× higher rollout throughput on Qwen3-32B by scaling to larger batch sizes (Figure 2, §3); (2) fused NF12 achieves up to 1.11× end-to-end single-sequence speedup, scaling with model size (§3); (3) NF12 decode is 2.85× faster than DFloat11, fast enough that the buffered path does not bottleneck compute-bound GEMM (§3). NF12 is lossless by construction (§2.1). The design lesson is counterintuitive: on GPU, a fixed-length code with SIMT-friendly decode can outperform entropy-optimal variable-length codes, even at a higher bit rate.

2 NYOOMFLOAT12

A practical format must be (a) fixed-length, so all threads in a warp decode identical data in lockstep; (b) lossless, since precision mismatch destabilizes RL training; and (c) register-only, since lookup tables that accelerate CPU decoders do not help on GPU. We describe how NF12 meets each: per-bit exponent structure enables fixed-length encoding (§2.1) with lossless output in 13 register operations, and two deployment modes target both regimes (§2.2). We then show that NF12 improves rollout throughput (§3), single-sequence inference (§3), and standalone decode speed (§3).

2.1 FORMAT

Trained neural network weights exhibit low exponent entropy: >99% of BF16 (Wang & Kanwar, 2019) weights cluster in exponents 112–127 across all models we tested (Appendix A). But this entropy is not uniform across bits. As Figure 1 shows, bits E7–E4 (the upper exponent nibble) are nearly constant at `0111`, with per-bit entropy $H < 0.02$. Bits E3–E0 retain high entropy ($H \approx 0.7$–1.0), indistinguishable from the mantissa. This per-bit structure, invisible to whole-field encoding, is what enables a fixed-length format.

NF12 stores 12 bits per weight: a 4-bit meta nibble (sign + E3, E2, E1) and the unchanged low byte (E0 + 7 mantissa bits). Eight weights go from 16 bytes to 12 (one meta byte per pair plus the unchanged lo bytes) for exactly $16/12 = 1.33\times$ compression. The single-weight encode/decode and register-level implementations for 2 and 8 weights are in Appendix D.

Decode and Escape Handling.

Decode restores the original BF16 value in 13 operations for 8 weights: 2 shifts, 5 LOP3 (fused ternary logic, via the PTX `lop3` instruction (NVIDIA)), and 6 PRMT (byte permutation; single-cycle data movement on NVIDIA GPUs). The final step ORs the constant `0x38003800` (the byte `0b00111000` repeated) to reconstruct the `0111` exponent prefix. All threads execute identical operations in lockstep.

The <0.5% of weights with exponents outside [112,127] (typically ∼0.15%; Appendix A) are handled via dense-sparse decomposition. Any group of 8 weights containing at least one out-of-range value is flagged by setting all meta nibbles to `0xF`; the rare in-range groups whose encoded meta happens to be all-`F` are treated identically, at negligible cost to compression ratio. Bulk decode produces zeros for flagged groups. The sparse component stores the 8 original BF16 values for each such group. In decode-to-memory, the originals overwrite the zeros, producing bitwise-exact output with no special handling in the decode hot path. In fused GEMV, the dense path contributes zero for escape positions and the sparse path accumulates the exact original values, computing the same dot-product terms. Storage overhead is under 1.5% (Appendix B).

Correctness. NF12 is lossless by construction. Bulk decode restores the constant upper exponent nibble for all in-range weights. Escape groups (meta = `0xFF`) decode to zero and are overwritten with the stored original BF16 values, producing bitwise-identical output.

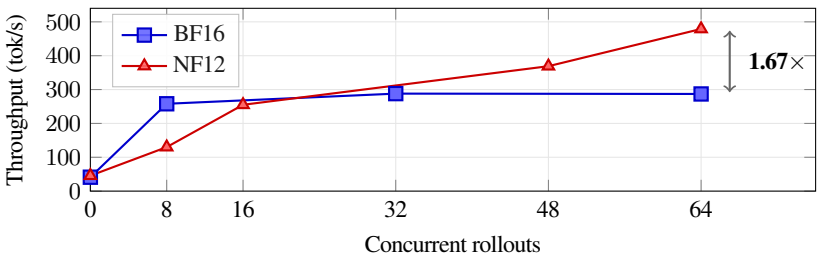

Figure 2: Throughput on Qwen3-32B (H100 80 GB, SGLang). NF12 is 1.11× faster at single-sequence inference, and the advantage grows with concurrent rollouts as compression frees memory for larger batches.

2.2 Two Deployment Modes

Fused (bandwidth-bound). Decode happens inside the GEMV inner loop. Each thread loads 12 compressed bytes, executes 13 operations, and produces 8 BF16 values directly in registers. Decoded weights never touch HBM or shared memory. This reduces HBM traffic at the cost of additional register-level arithmetic, analogous in spirit to how FlashAttention (Dao et al., 2022) trades compute for reduced memory movement in attention.

Buffered (capacity-bound). Weights remain compressed in HBM. Before each layer's GEMM, a decode kernel reads the NF12 weights and writes BF16 to a reusable buffer. NF12 decode is fast (744 Gelem/s to HBM, §3), but the buffered path adds memory movement compared to BF16: the decompressed layer must be written to and then read from the buffer. At high batch sizes the GEMM is compute-bound, so this extra memory movement is not on the critical path. Only one layer's buffer (∼0.5 GB for a 32B model's largest layer) is live at a time.

The benefit is memory capacity. For Qwen3-32B on 80 GB H100: BF16 weights occupy ∼64 GB, leaving ∼16 GB for KV caches and activations. NF12 weights occupy ∼48 GB plus a ∼0.5 GB reusable buffer, leaving ∼31.5 GB, roughly 2× the headroom. This translates directly to batch capacity: more room for KV caches means more concurrent sequences.

3 Experiments

For NF12 to be useful, it must be fast enough to improve throughput in both the bandwidth-bound and capacity-bound regimes. NF12 is lossless by construction (§2.1); the remaining question is whether decode overhead offsets the bandwidth and capacity savings. We evaluate: (1) does compression improve rollout throughput at scale? (2) Does fused decode speed up single-sequence inference? (3) Is standalone decode fast enough for the buffered path?

Setup. NVIDIA H100 80 GB SXM (3.35 TB/s HBM3), CUDA 12.x, PyTorch 2.9. Real Llama and Qwen weights from HuggingFace. GEMV benchmarks: 10 runs × 500 iterations, L2 cache flushed between iterations to measure true HBM bandwidth, CUDA events timing. Baselines: (1) cuBLASLt with `CUDA_R_16BF` and heuristic tuning; (2) hand-tuned BF16 GEMV kernel (same algorithmic structure as the NF12 kernel, minus decode) achieving 48–92% memory bandwidth utilization (MBU). Rollout experiments (§3): SGLang with custom NF12 weight loader; weights decoded layer-by-layer via dedicated CUDA kernel before each layer's GEMM.

Rollout Throughput. NF12 reaches **1.67**× higher rollout throughput on Qwen3-32B by freeing memory for larger batches (Figure 2). BF16 throughput plateaus at ∼288 tok/s by batch 32 as memory pressure forces request queuing; batch 64 completes in the same time as two passes of batch 32. NF12's 25% smaller weight footprint frees capacity for more concurrent sequences, and throughput continues scaling to 479 tok/s at batch 64. The layer-by-layer decode adds memory movement, visible at batch 8 where NF12 is 2× slower; by batch 16 the capacity advantage dominates, and the benefit grows with batch size.

Single-Sequence Inference. Fused NF12 achieves up to 1.11× end-to-end speedup on single-sequence inference (Table 1), relevant for serving after RL training. In fused mode, decode happens inside the GEMV inner loop with no HBM round-trip; token generation at batch size 1 is memory-bandwidth bound, so reading 25% fewer bytes translates directly to speedup.

Table 1: End-to-end single-sequence inference (tok/s, batch size 1). Fused NF12 decode in GEMV loop.

Model	BF16	NF12	Speedup
Llama-1B	606.8	575.0	0.95×
Llama-8B	152.3	160.2	**1.05**×
Qwen3-14B	87.4	95.6	**1.09**×
Qwen3-32B	41.1	45.6	**1.11**×

Table 2: Standalone decode bandwidth (512M elements, L2 flushed). NF12 decode is near HBM limits at all three destinations.

Operation	GB/s
H100 peak read	3,350
BF16 copy (read+write)	2,780
NF12 decode-to-registers (read)	2,927
NF12 decode-to-SMEM (read)	1,205
NF12 decode-to-HBM (read+write)	2,604
DFloat11 decode-to-HBM (read+write)	881
nvCOMP GDeflate (read+write)	24

Speedup scales with model size: from near-neutral on Llama-1B (0.95×) to **1.11**× on Qwen3-32B, as compressed linear layers become a larger fraction of total inference time. Per-layer GEMV microbenchmarks reach 1.13×–1.28× (Appendix C), approaching the $16/12 = 1.33\times$ theoretical maximum; the end-to-end speedup is lower because attention, normalization, and sampling do not benefit from weight compression.

Standalone Decode Performance. The fused path achieves 87% of peak HBM read; the buffered path achieves 2,604 GB/s, comparable to BF16 copy despite decoding (Table 2). NF12 is **2.85**×faster than DFloat11 (881 GB/s) and two orders of magnitude faster than general-purpose GPU compression (24 GB/s).

4 Related Work

Lossless weight compression. DFloat11 (Zhang et al., 2025) Huffman-codes BF16 exponents (∼11 bits, 31% compression); ECF8 (Yang et al., 2025) extends to FP8. Both treat decode as preprocessing, not fusing it into compute. HuffLLM (Yubeaton et al., 2025) attempts kernel fusion but faces variable-length parallelism challenges. NF12 trades ∼1 bit per weight for fixed-length decode that eliminates warp divergence, enables register-level fusion into GEMV, and achieves 2.85× faster decode (§3).

Lossy quantization. GPTQ (Frantar et al., 2022), AWQ (Lin et al., 2024), SmoothQuant (Xiao et al., 2023), and QLoRA (Dettmers et al., 2023) reduce precision for speedup but are lossy. NF12 targets BF16 specifically; quantization to narrower formats removes the exponent redundancy NF12 exploits, though lossless compression of quantized formats is possible (e.g., ECF8 (Yang et al., 2025) extends DFloat11 to FP8). For RL post-training, Xi et al. (2026) show that FP8 rollout generation introduces a precision mismatch that causes training divergence on long sequences; lossless compression avoids this failure mode entirely.

IO-aware kernel design. FlashAttention (Dao et al., 2022) reduces HBM traffic for attention via tiling and fusion; Alted showed compression can accelerate memory-bound CPU workloads (Alted, 2010). NF12's fused mode is inspired by the same principle: trade arithmetic (decode) for reduced HBM reads.

5 Conclusion

We presented NyoomFloat12, a lossless 12-bit format that addresses GPU memory capacity constraints in RL post-training without the precision mismatch that makes FP8 rollouts unstable (Xi et al., 2026). The key insight, that the upper exponent nibble is nearly constant across trained weights, enables fixed-length decode with 13 SIMT operations whose overhead is low enough to hide behind compute-bound GEMM. The 25% memory reduction frees capacity for larger batches, yielding 1.67× higher rollout throughput on Qwen3-32B; in the bandwidth-bound regime, fused decode achieves up to 1.11× end-to-end single-sequence speedup. Fixed-length codes with SIMT-friendly bitwise decode can outperform entropy-optimal codes on GPU, even when storing more bits per weight.

REFERENCES

Francesc Alted. Why modern cpus are starving and what can be done about it. *Computing in Science & Engineering*, 12(2):68–71, 2010.

Tri Dao, Dan Fu, Stefano Ermon, Atri Rudra, and Christopher Ré. Flashattention: Fast and memory-efficient exact attention with io-awareness. *Advances in neural information processing systems*, 35: 16344–16359, 2022.

DeepSeek-AI. Deepseek-r1: Incentivizing reasoning capability in LLMs via reinforcement learning. *arXiv:2501.12948*, 2025.

Tim Dettmers, Artidoro Pagnoni, Ari Holtzman, and Luke Zettlemoyer. Qlora: Efficient finetuning of quantized llms. *Advances in neural information processing systems*, 36:10088–10115, 2023.

Elias Frantar, Saleh Ashkboos, Torsten Hoefler, and Dan Alistarh. Gptq: Accurate post-training quantization for generative pre-trained transformers. *arXiv preprint arXiv:2210.17323*, 2022.

Woosuk Kwon, Zhuohan Li, Siyuan Zhuang, Ying Sheng, Lianmin Zheng, Cody Hao Yu, Joseph E. Gonzalez, Hao Zhang, and Ion Stoica. Efficient memory management for large language model serving with PagedAttention. In *Proceedings of the 29th Symposium on Operating Systems Principles*, pp. 611–626, 2023.

Ji Lin, Jiaming Tang, Haotian Tang, Shang Yang, Wei-Ming Chen, Wei-Chen Wang, Guangxuan Xiao, Xingyu Dang, Chuang Gan, and Song Han. Awq: Activation-aware weight quantization for on-device llm compression and acceleration. *Proceedings of Machine Learning and Systems*, 6:87–100, 2024.

NVIDIA. Parallel thread execution isa. `https://docs.nvidia.com/cuda/parallel-thread-execution/`. Accessed 2026-02-06.

Pol G Recasens, Ferran Agullo, Yue Zhu, Chen Wang, Eun Kyung Lee, Olivier Tardieu, Jordi Torres, and Josep Ll Berral. Mind the memory gap: Unveiling gpu bottlenecks in large-batch llm inference. *arXiv:2503.08311*, 2025.

Shibo Wang and Pankaj Kanwar. Bfloat16: The secret to high performance on cloud tpus. `https://cloud.google.com/blog/products/ai-machine-learning/bfloat16-the-secret-to-high-performance-on-cloud-tpus`, 2019. Accessed 2026-02-06.

Haocheng Xi, Charlie Ruan, Peiyuan Liao, Yujun Lin, Han Cai, Yilong Zhao, Shuo Yang, Kurt Keutzer, Song Han, and Ligeng Zhu. Jet-RL: Enabling on-policy FP8 reinforcement learning with unified training and rollout precision flow. *arXiv:2601.14243*, 2026.

Guangxuan Xiao, Ji Lin, Mickael Seznec, Hao Wu, Julien Demouth, and Song Han. Smoothquant: Accurate and efficient post-training quantization for large language models. In *International Conference on Machine Learning*, pp. 38087–38099. PMLR, 2023.

Zeyu Yang, Tianyi Zhang, Jianwen Xie, Chuan Li, Zhaozhuo Xu, and Anshumali Shrivastava. To compress or not? pushing the frontier of lossless genai model weights compression with exponent concentration. *arXiv:2510.02676*, 2025.

Patrick Yubeaton, Tareq Mahmoud, Shehab Naga, Pooria Taheri, Tianhua Xia, Arun George, Yasmein Khalil, Sai Qian Zhang, Siddharth Joshi, Chinmay Hegde, and Siddharth Garg. Huff-llm: End-to-end lossless compression for efficient llm inference. *arXiv:2502.00922*, 2025.

Tianyi Zhang, Mohsen Hariri, Shaochen Zhong, Vipin Chaudhary, Yang Sui, Xia Hu, and Anshumali Shrivastava. 70% size, 100% accuracy: Lossless LLM compression for efficient GPU inference via dynamic-length float (DFloat11). In *The Thirty-ninth Annual Conference on Neural Information Processing Systems*, 2025. URL `https://openreview.net/forum?id=xdNAVP7TGy`.

Lianmin Zheng et al. Sglang: Efficient execution of structured language model programs, 2023.

Table 3: Exponent concentration across 9 LLMs. H_{total}: sum of per-bit entropies (max 16). All models show >99% of weights in exponents 112–127.

Model	Params	Weights	Escape %	H_{total}
Qwen-0.5B	0.5B	314M	0.14	11.75
Llama-1B	1.2B	805M	0.14	11.78
Qwen-1.5B	1.5B	1.2B	0.09	11.51
Phi-2	2.7B	1.7B	0.13	11.49
Qwen-3B	3.0B	2.4B	0.29	11.73
Llama-3B	3.2B	2.1B	0.14	11.75
Qwen-7B	7.0B	5.7B	0.21	11.84
Llama-8B	8.0B	5.6B	0.20	11.94
Qwen-14B	14B	10.2B	0.52	11.76

Table 4: Exponent concentration beyond LLMs. All models show >99.7% in the compressible range.

Model	Type	Params	Escape %
DINOv2-B	Vision	86M	0.05
DINOv2-L	Vision	304M	0.06
SD 1.5	Diffusion	860M	0.03
SDXL	Diffusion	2.6B	0.03
Moonlight-16B	Muon opt.	16B	0.03

A Exponent Concentration

The pattern holds across architectures and training methods:

Exponent concentration is a property of trained weights broadly, not specific to LLMs or AdamW.

B Escape Statistics

On Llama-3.2-3B (3.2B weights): 4,380,769 escapes (0.136%), maximum 48 escapes per row, zero rows exceeding 256 escapes. Dense storage: 4.82 GB (12 b/w). Escape storage (full groups of 8 original BF16 values): 17.5 MB. Effective: 12.04 bits/weight (+0.3% overhead).

C Full GEMV Results

Table 5 shows per-layer GEMV performance on both `down_proj` and `gate_up` MLP layers. Speedup reaches 1.28× on the largest shapes, approaching the $16/12 = 1.33\times$ bandwidth-reduction ceiling.

D Register-Level Encode/Decode

Algorithm 1 shows the single-weight encode/decode. Algorithm 2 shows encode/decode for two packed BF16 values in a single 32-bit register (the unit stored on disk). Algorithm 3 shows the full 8-weight bulk decode used in the GEMV inner loop, derived from the CUDA implementation. The 13-op count (2 shifts, 5 fused ternary logic, 6 byte selects) is verified via `cuobjdump --dump-sass`.

Each output register holds two bf16 values: `[hi, lo, hi, lo]`. Step 2 uses single-cycle byte-select instructions (`__byte_perm` on NVIDIA GPUs) to pick one byte from each source, assembling the output in 6 instructions. The entire 8-weight decode executes in 13 instructions with no control flow or shared memory access.

Table 5: Full GEMV performance (μs $\pm$ std). 10 runs $\times$ 500 iterations, L2 flushed.

Model	Layer	K×N	cuBLAS	BF16	NF12	×bf16
Llama-1B	down	8k×2k	26.7	20.9	18.5	**1.13**
Llama-1B	gate_up	2k×8k	37.2	34.1	32.5	1.05
Llama-3B	down	8k×3k	31.2	27.3	24.0	**1.14**
Llama-3B	gate_up	3k×8k	49.4	46.4	40.0	**1.16**
Llama-8B	down	14k×4k	55.0	51.6	44.3	**1.16**
Llama-8B	gate_up	4k×14k	92.1	87.3	70.3	**1.24**
Llama-70B	down	29k×8k	166.2	162.8	134.2	**1.21**
Llama-70B	gate_up	8k×29k	324.5	305.1	239.2	**1.28**
Qwen3-8B	down	12k×4k	49.7	46.2	39.8	**1.16**
Qwen3-8B	gate_up	4k×12k	84.0	77.1	62.7	**1.23**
Qwen3-14B	down	17k×5k	75.1	72.2	60.1	**1.20**
Qwen3-14B	gate_up	5k×17k	135.3	125.6	99.8	**1.26**
Qwen3-32B	down	26k×5k	100.5	98.1	81.1	**1.21**
Qwen3-32B	gate_up	5k×26k	191.9	177.5	139.2	**1.28**

Algorithm 1 NF12 Single-Weight Encode/Decode

procedure ENCODE(x : bf16) $\rightarrow$ uint12
  return $(x \;\&\; \texttt{0x8000}) \gg 4$ {sign $\rightarrow$ bit 11}
    $\mid (x \;\&\; \texttt{0x0700})$ {E3:E1 stay at bits 10:8}
    $\mid (x \;\&\; \texttt{0x00FF})$ {lo byte unchanged}

procedure DECODE(y : uint12) $\rightarrow$ bf16
  return $(y \;\&\; \texttt{0x0800}) \ll 4$ {sign $\rightarrow$ bit 15}
    $\mid$ `0x3800` {inject `0111` prefix}
    $\mid (y \;\&\; \texttt{0x07FF})$ {E3:E0 + mantissa}

Algorithm 2 NF12 Encode/Decode for 2 Packed BF16 Weights

procedure ENCODE(x : bf16×2 as uint32) $\rightarrow$ uint24
  $y \leftarrow x \;\&\; \texttt{0x00FF87FF}$ {keep lo bytes + B's meta}
  $y \leftarrow y \mid ((x \;\&\; \texttt{0x80000000}) \gg 17)$ {A's sign}
  $y \leftarrow y \mid ((x \;\&\; \texttt{0x07000000}) \gg 13)$ {A's E3:E1}
  return y

procedure DECODE(y : uint24) $\rightarrow$ bf16×2 as uint32
  $x \leftarrow y \;\&\; \texttt{0x00FF87FF}$
  $x \leftarrow x \mid ((y \;\&\; \texttt{0x00004000}) \ll 17)$ {A's sign}
  $x \leftarrow x \mid ((y \;\&\; \texttt{0x00003800}) \ll 13)$ {A's E3:E1}
  $x \leftarrow x \mid$ `0x38003800` {inject `0111` prefix}
  return x

Algorithm 3 NF12 Bulk Decode: 8 Weights in 13 Ops

Require: M: 4 meta bytes (uint32), L[8]: 8 lo bytes (as two uint32s L_0,L_1)
Ensure: O_0,O_1,O_2,O_3: 4 registers $\times$ 2 bf16 values
{Step 1: Build hi bytes from meta (2 shifts + 5 ternary logic ops)}
$odd \leftarrow (M \;\&\; \texttt{0x87878787}) \mid \texttt{0x38383838}$
$even \leftarrow ((M \ll 1) \;\&\; \texttt{0x80808080}) \mid ((M \gg 3) \;\&\; \texttt{0x07070707}) \mid \texttt{0x38383838}$
{Step 2: Assemble bf16 pairs (hi byte from meta, lo byte from L)}
$O_0 \leftarrow [even[0], L[0], odd[0], L[1]]$ {weights 0, 1}
$O_1 \leftarrow [even[1], L[2], odd[1], L[3]]$ {weights 2, 3}
$O_2 \leftarrow [even[2], L[4], odd[2], L[5]]$ {weights 4, 5}
$O_3 \leftarrow [even[3], L[6], odd[3], L[7]]$ {weights 6, 7}