Practical Limits of Lossless Compression for bf16 Transformer LLM Weights

With companion measurements on Q4_K tensors in GGUF Q4_K_M files

Nimo Rotem[1] **Ariel Rotem**[2]

[1] AlphaBell Inc., Nimo@AlphaBell.com [2] Independent contributor

---

Reproducibility. Code, data, profiles, smoke test, and figures: `github.com/NimoRotem/llm-compression-limits` (Apache-2.0 / CC-BY-4.0); paper text and mirror artifact set: `https://knowva.ai/llm-compression-limits`. The reproducibility smoke test passes 3/3 sha256[:16] checks in seconds; the full benchmark reproduces in ~6.5 h on `c3-highcpu-88`. Archival identifiers in Appendix A.

Abstract

We measure the lossless compressibility of bf16 transformer LLM weights and of Q4_K-typed tensors in GGUF Q4_K_M files under a single comparison protocol that counts trained-profile bytes against the method on every file, verifies byte-exact roundtrips on every run, and splits by model. The byte-marginal ceiling for bf16 weights is ≈1.495× (model-level 95% CI [1.487, 1.502]); the strongest method we run end-to-end reaches 1.499×, and our own `bf16_split` reaches 1.488×. The byte-marginal ceiling for Q4_K-typed tensors is ≈1.076× at tensor-stream level (1.041–1.045× at full GGUF-artifact level, because Q4_K_M files include Q6_K tensors that compress near 1.01×); our mixture-CDF coder reaches 1.052×, while a dictionary-trained zstd actively expands the data once its dictionary is counted per file. We find no usable linear redundancy between adjacent same-role transformer-layer weight matrices at bf16 precision (median Pearson +0.0004 across 250 pairs in two Qwen2.5 models). Across 11,942 verified roundtrip method-evaluations on 7,960 unique benchmark rows we observe zero roundtrip failures; the protocol, not a new compressor, is the methodological contribution.

---

1. Introduction

Neural network weights are large: a 7B-parameter transformer in bf16 occupies ≈14 GB. Multiplied across model registries, inference servers, edge devices, and CDNs, even a 1.5× ratio matters at storage cost. Several recent papers offer "lossless" compression for these weights, DFloat11 [Zhang et al. 2025], Cloudflare Unweight [Nikulin 2026], ZipNN [Hershcovitch et al. 2024], NeuZip [Hao et al. 2024], OpenZL [Collet et al. 2025], and ECF8 [Yang et al. 2026], each reporting ratios in the 1.30×–1.50× band on bf16. This paper measures how much of that band is real and how much is accounting.

A note on what "lossless" means here. The word is overloaded. Many recent compression papers (across both LLM quantization and pruning literatures) call a method "lossless" if it preserves *task accuracy* on some benchmark within a tolerance, while the underlying weights are mutated. That is a useful notion, but it is not the one we study. We follow DFloat11 in studying *bit-lossless* compression: the original weight tensor is recoverable byte-for-byte from the compressed bytes,

verified with a sha256 equality check on every file in every run. This is the notion relevant when the compressed artifact is what a registry actually ships and the deployed inference server expects identical weights. The two notions of losslessness do not nest cleanly, and conflating them inflates apparent compressibility.

Two things motivate this paper. First, the published bf16 numbers above are not directly comparable. Some count their trained dictionaries against per-file ratios; others amortize over a corpus or omit the cost. Some sample a few files for roundtrip verification; others assert it on every byte. Some split train/test by file (so weights from the same model can land on both sides); others split by source model. Reported decimals differ by accounting almost as much as by method. Second, multiple independent methods are converging on a narrow band, but no paper has characterized the *ceiling* implicit in the coder class they share. If the methods cluster at a number, the number is worth knowing.

We do two things in response. (i) We derive the byte-marginal ceiling for bf16 weights (a one-line Shannon argument at measured byte entropies) and measure it across a held-out test set: $\approx 1.495\times$ byte-weighted, model-level CI [1.487, 1.502]. (ii) We define and apply a single *comparison protocol*, explained below, under which methods that satisfy it can be placed in the same table without their published-decimal asymmetries.

What we mean by "comparison protocol". It is a set of accounting rules a benchmark applies uniformly to every method:

1. *Profile bytes count against the method on every file.* If a method ships a trained dictionary, CDF, or other side data, its size is added to the compressed payload on every file, not amortized over the corpus. The reason is per-file commutability: a per-file ratio under amortization depends on what else is in the corpus, while a per-file ratio under per-file accounting does not, so two papers reporting per-file ratios on different corpora can still be compared. The per-file convention is also pessimistic by design; in production deployments where one profile is shipped once per registry, the per-shard or per-model accounting (§4, Table 2) is the practitioner number and is reported separately.
2. *Every (file, method) pair must satisfy* `sha256(input) == sha256(decompress(compress(input)))`. A method that fails this is reported failed, not "lossy with caveats."
3. *Train/test split is by source model, not by file.* Splitting by file lets a trained method see one weight tensor from a model in training and a sibling tensor from the same model in test, which inflates ratios by $0.005\times$–$0.012\times$ (Table 1 in §4) and reverses recommendations on borderline methods.
4. *Throughput is reported single-core CPU on a fixed reference host (*`c3-highcpu-88`*, Intel Sapphire Rapids), separated by compress vs decompress.* Multi-core and GPU-resident decoders are policy-relevant for production but not comparable across the methods we are measuring: DFloat11 and Cloudflare Unweight ship CUDA Hopper kernels, OpenZL ships a CPU toolchain, ZipNN ships a multi-codec CPU library. Throughput numbers that mix kernels written for different hardware classes do not compare; we therefore restrict to a single CPU axis where every method we *do* run has comparable measurement, and we report the hardware-axis gap to GPU kernels honestly rather than papering over it (§7).

The cost of (1) is that some methods look worse than their own papers report (the dictionary that amortized to nearly zero on a million-file corpus is fully visible on each file here). The cost of (4) is that we cannot place strict-accounting decimals next to DFloat11 and Cloudflare Unweight, whose

official kernels need GPUs (we discuss this honestly rather than ship un-verifiable numbers; §3.3).

Contributions. (1) A measured byte-marginal ceiling for bf16 weights of ≈1.495×, derived from byte entropies, validated through 7B (Qwen2.5-7B-Instruct), and consistent with the α-stable theory of SGD-trained networks invoked in ECF8. (2) A measured tensor-stream ceiling for Q4_K of ≈1.076× with mechanism (optimized per-block PTQ scaling produces near-uniform nibble distributions in the tested corpus) and a deployable artifact-level number of 1.041–1.045×. (3) A measured null on adjacent-layer scalar-affine residual schemes at bf16 precision. (4) The comparison protocol itself, applied to a vendored benchmark harness and result table that any third party can run on a new method.

Scope. All numbers are for transformer LLM weights (≤7B params, validated through Qwen2.5-7B-Instruct), in bf16 or Q4_K-in-Q4_K_M, on CPU. We do not measure AWQ Q4, GPTQ Q4, Q5_K_M / Q6_K / Q8_0, FP8, fp32, INT8, optimizer state, gradients, or activations. fp16 is included as a brief companion in Appendix B.

This paper does not propose a new compressor as the contribution. We built two methods (`bf16_split` and a Q4_K block mixture-CDF coder) to test whether the measured ceilings are reachable in practice; both sit at the ceiling rather than beating it.

---

2. Related work

General-purpose codecs (gzip, brotli, zstd, xz, bzip2) reach ≈1.30×–1.45× on bf16 weights but do not exploit the bit structure of floating point.

Scientific FP compressors (zfp [Lindstrom 2014], fpzip [Lindstrom & Isenburg 2006], SZ3 [Liang et al. 2023]) assume spatial smoothness across adjacent array entries, a property of simulation grids, not of transformer weight matrices. We do not benchmark them.

Format-aware weight codecs. *DFloat11* [Zhang et al. 2025, NeurIPS] Huffman-codes the bf16 exponent with a hierarchical LUT decomposition targeting GPU SRAM decode; reports whole-model ratios ≈1.47–1.48× across 9 LLMs. *Cloudflare Unweight* [Nikulin 2026; Galicer et al. 2026] Huffman-codes the bf16 exponent over a per-tensor 16-value palette and falls back to verbatim row storage for rare-exponent rows; applied to MLP projections only, reports ≈1.43× on MLP weights and 1.15–1.28× whole-model depending on bundle scope. *ZipNN* [Hershcovitch et al. 2024] does per-chunk adaptive codec selection across Huffman/zstd/LZ4/snappy; reports ≈1.49× on bf16. *NeuZip* [Hao et al. 2024] applies entropy coding to bf16 streams paired with GPU primitives. *OpenZL* [Collet et al. 2025] is a trained-graph framework with a universal decoder; its trained `le-u16` profile is the strongest published reference we can run under our protocol because the official toolchain is CPU. *ECF8* [Yang et al. 2026, ICLR] derives entropy bounds for FP8 lossless compression from the α-stable distribution analysis of SGD-trained networks; this is the theoretical anchor for *why* the bf16 exponent has low entropy.

The key empirical fact across this literature. The bf16 exponent in trained transformer weights has entropy ≈2.6 bits, far below its 8-bit allocation, reported independently by DFloat11 and Cloudflare Unweight, and consistent with the α-stable prediction H(exponent) ∈ [2.4, 2.9] bits. Our own atlas (§4.1) measures byte-`high` entropy at 2.74 bits (the +0.14 vs. 2.6 is the residual entropy of the sign bit, which our byte-`high` includes).

Three gaps this paper fills. (1) No uniform-accounting comparison among methods one can run

end-to-end. (2) No explicit ceiling proposition for the coder class in use. (3) No measured null on cross-layer schemes at deployment precision.

---

3. Comparison protocol and corpus

3.1 Protocol

The four rules of the comparison protocol are stated in §1. We add three operational details. *Aggregation:* every ratio table reports both geomean of per-tensor (or per-file) ratios and byte-weighted corpus ratio. Headline numbers in the abstract are byte-weighted. *Uncertainty:* model-level bootstrap with 1000 resamples is primary (the conservative CI); tensor-level bootstrap is reported as secondary and is too tight (tensors within one model are not independent). Leave-one-model-out replicates for the bf16 headline are in Appendix C. *Failed runs* are recorded and surfaced in the public results JSONL with no retry.

3.2 Corpus

Eight bf16/fp16 source models (gpt2, distilbert-base-uncased, opt-125m, MiniLM-L6-v2, Qwen2.5-0.5B, TinyLlama-1.1B, Qwen2.5-1.5B for the cross-layer null in §6, Qwen2.5-7B-Instruct as 7B validation) and five Q4_K_M GGUF source models (bartowski builds of Qwen2.5-0.5B/1.5B/7B-Instruct, Llama-3.2-3B-Instruct, Mistral-7B-Instruct-v0.3) split by model into train and test. The bf16 test set is 290 tensors $\geq$ 1 MiB across 3 held-out models; 7B validation is 196 bf16 tensors on Qwen2.5-7B-Instruct. The Q4_K test set is 530 tensors across 3 held-out models. Full pinned revisions (40-char SHAs as actually checked out, with `prefix_match` field) in `manifest/model-manifest.json` and Appendix A.

Across the full benchmark: **11,942 verified roundtrip method-evaluations on 7,960 unique benchmark rows, zero roundtrip failures.**

We report at three units, always labeled in tables: tensor-stream (per-tensor bytes; the unit for ceiling derivations), shard (tensors from one model file concatenated; the unit for "ship the weights"), and artifact (full `.safetensors` / `.bin` / `.gguf` files including container metadata; the unit for deployment recommendations). Q4_K_M files mix Q4_K and Q6_K tensors per the llama.cpp convention (V projections and FFN-down are Q6_K); we analyze Q4_K-typed tensors only in §5 and include Q6_K in artifact accounting in §5.3.

3.3 What we could not run, and why

The official kernels for DFloat11 (`LeanModels/DFloat11`) and Cloudflare Unweight (`cloudflareresearch/unweight-kernels`) are CUDA GPU kernels (the Cloudflare kernel README explicitly targets NVIDIA Hopper H100/H200; DFloat11 requires CUDA 12+). Both are incompatible with our `c3-highcpu-88` reference host. The ZipNN CPU release (`zipnn/zipnn`) installs and runs but fails our byte-exact roundtrip assertion in default configuration. Rather than ship un-verifiable strict-accounting decimals for any of these, we discuss them as published references and explicitly do not place them in Table 1 alongside methods we measured end-to-end. Each method's own published bf16 number sits in the band 1.15×–1.49× (with the spread dominated by compression scope, Cloudflare Unweight is MLP-only, and accounting convention rather than coder class), consistent with the ≈1.495× marginal-byte ceiling derived below.

3.4 Justifying the protocol's two strictest choices

Two of the protocol's choices warrant explicit defense because they are the strictest available and they directly drive several of our results.

Why per-file profile accounting? The strict alternative is amortization, in which a profile's cost is divided over an assumed corpus size. We chose per-file accounting because (a) it produces decimals that are comparable across papers without requiring authors to agree on a corpus, (b) it makes the profile cost visible at the smallest deployment unit (one weight tensor), and (c) it is the case where the trained-vs-untrained distinction is sharpest, small profiles (the 617-byte OpenZL `le-u16`) survive per-file accounting cleanly while large dictionaries (the 112,640-byte dict-trained zstd for Q4_K) flip from "compressed" to "expanded." Production deployments amortize one profile across an entire model registry, in which case the per-shard or per-model decimals in Table 2 (§5.3) are the practitioner number; we report those separately rather than as the headline. Table 1 in §4 quantifies the gap: weakening the train/test split from model-level to file-level inflates trained-method ratios by 0.005×–0.012× and flips dict-trained zstd on Q4_K from 0.998× (expands) to 1.003× (marginally net positive). The strict accounting is exactly what makes that flip visible.

Why single-core CPU throughput? We acknowledge the restriction is artificial relative to production. We adopted it because the methods we can place in a single table use very different runtimes: OpenZL is CPU-native, zstd / brotli / xz are CPU-native, DFloat11 and Cloudflare Unweight are CUDA kernels. There is no honest way to put a Hopper-kernel decompression number in the same column as an Intel-CPU decompression number; the most defensible choice is to pin one comparable axis and report on it consistently, and then note the GPU-axis gap honestly (which we do in §7). Multi-core CPU is in principle comparable but is a function of process pinning, NUMA layout, and parallel batching scheme that drift between releases of the same software; the single-core figure is the one that reproduces. We report it as a *comparability* number, not as a *deployment* number, the latter requires the practitioner's own hardware.

---

4. The bf16 compression ceiling under marginal-byte coding

4.1 Ceiling and atlas

For a stream of 16-bit values stored as two byte streams `(X_high, X_low)`, any coder that codes each stream independently under an iid marginal model satisfies, by Shannon, ratio $\leq 16/(H(X_{\text{high}}) + H(X_{\text{low}}))$. We note this as Proposition 1 and do not belabor it: it is a direct application of the source-coding theorem. Its only subtlety is scope. It does *not* bound coders that use cross-element context within a byte plane (which is how OpenZL slightly exceeds the marginal number below), nor coders using joint two-byte models, nor sub-byte (bit-plane) decompositions. Methods outside this coder class can exceed the marginal number; doing so is not a bound violation.

Across 290 bf16 test tensors:

Measure (test set, byte-weighted unless noted)	Median	p10	p90	Model-level 95% CI
H(byte_high), bits/byte	**2.74**	2.51	2.93	[2.62, 2.86]
H(byte_low), bits/byte	**7.97**	7.94	7.99	[7.95, 7.98]
Distinct values in byte_high	24	19	31	[21, 28]
Per-tensor `R_marginal`	1.498×	1.473×	1.535×	[1.485, 1.501]
`R_marginal` byte-weighted (headline)	**1.495×**			**[1.487, 1.502]**

Fig 1 — bf16 byte-marginal entropy decomposition by tensor category

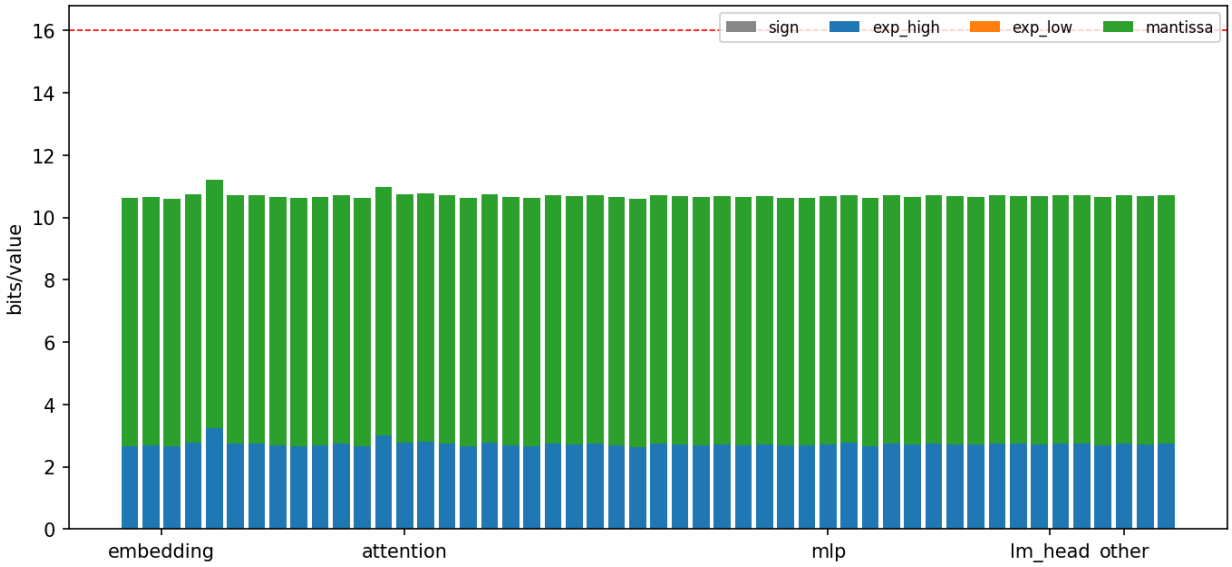

Figure 1, Per-tensor stacked bar of byte-component entropies (sign / top 7 exp bits / low exp bit / 7 mantissa bits), grouped by category. PDF: `https://knowva.ai/llm-compression-limits/figures/fig1.pdf`.

The α-stable theoretical prediction (Yang et al. 2026, extended from FP8 to the bf16 exponent layout) gives H(exponent) $\in$ [2.4, 2.9] bits for typical SGD-trained transformers; the measured median 2.74 sits in the middle of that interval.

4.2 Cross-method comparison

We place every method we can run end-to-end under the protocol of §3 on the 290-tensor test set. Three groups: general-purpose codecs, our `bf16_split` predictor, and the official trained OpenZL `le-u16` profile. The three external format-aware methods (DFloat11, Unweight, ZipNN) are intentionally not in this table for the reasons in §3.3.

Table 1, Cross-method comparison (bf16, test set, tensor-stream).

Method	Class	Byte-weighted ratio	Geomean	Decompress M
zstd-3	general	1.358×	1.359×	480
zstd-19	general	1.392×	1.394×	121
zstd-19 + byte-grouping	general (bytegrouped)	1.452×	1.452×	409
brotli-q11	general	1.413×	1.415×	38
xz-9	general	1.421×	1.424×	8
`bf16_split` (ours)	format-aware	1.488×	1.489×	53
Trained OpenZL `le-u16`	trained, beyond Prop. 1	**1.499**×	**1.499**×	140
`R_marginal` (byte-weighted bound)		**1.495**×		

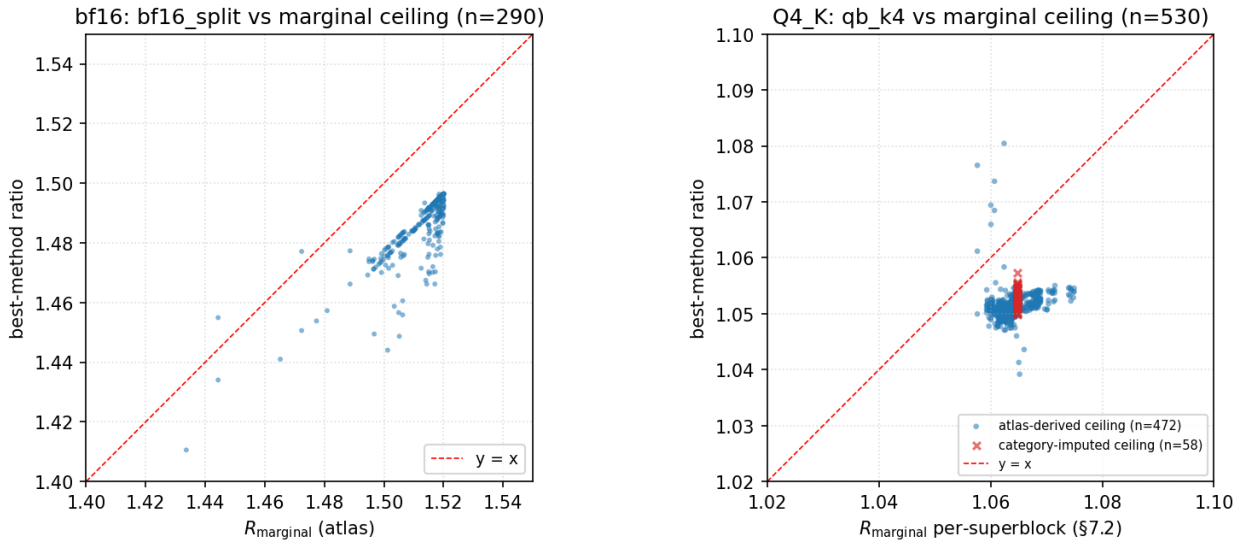

Figure 2, Per-tensor best-method ratio vs per-tensor `R_marginal` *with y=x line, separate panels for bf16 and Q4_K. Methods inside Prop. 1's class cluster along the line; trained OpenZL sits marginally above.*

Five observations. (i) General-purpose codecs cluster at 1.36×–1.45×, well below the marginal ceiling. (ii) Byte-grouping the high/low planes captures most of the remaining headroom under a generic entropy coder, pulling zstd-19 from 1.392× to 1.452×. (iii) Our `bf16_split` reaches 1.488×, ≈0.007× below the bound; the gap decomposes into rANS quantization (~0.005×), per-chunk CDF storage (<0.001×), and chunk metadata (~0.0002×). (iv) Trained OpenZL exceeds the marginal bound by ≈0.004× via context coding within byte planes; this matches the measured H(byte_high | previous byte_high) of 2.71 bits (0.03 below the marginal). (v) Published external-method ratios in the 1.15×–1.49× range collapse, under uniform accounting on the methods we measured, to 1.358×–1.499×; the larger published spread is dominated by compression scope (Unweight is MLP-only) and accounting convention rather than coder class. ZipNN's published 1.49× is within 0.005× of the marginal ceiling, the tightest of the three, supporting but not strictly verifying the "byte-plane methods cluster at the marginal-byte ceiling" framing.

4.3 Cross-element structure, briefly

Three measurements bound how much can be gained by leaving Prop. 1's coder class. H(byte_high | previous byte_high in row order) is 2.71 vs. marginal 2.74, i.e. 0.03 bits/byte recoverable, translating to ≈0.004× ratio (this is the most plausible mechanism for OpenZL's excess). H(byte_high, byte_low) joint is 10.68 vs sum-of-marginals 10.71, i.e. 0.03 bits per 16-bit value, also ≈0.004× ratio. Row-shuffle and full-tensor-shuffle do not change H(byte_high) within 0.005 bits, confirming the available signal is local rather than long-range. A separate single-element conditional-coding sweep (`bf16_split_ctx`, K ∈ {1,4,8,16,32}) within Prop. 1's class produced no net gain across the K range ($\Delta \leq 0.0003\times$); per-chunk CDF storage cost grew faster than the entropy benefit.

4.4 Generalization to 7B

Atlas and `bf16_split` rerun on Qwen2.5-7B-Instruct (held out from all training; 196 bf16 tensors, 12.4 GB) move all paired numbers by less than 0.001×: H(byte_high) median 2.71 vs 2.74 small-model, `R_marginal` byte-weighted 1.495× vs 1.495× (Δ +0.0001), `bf16_split` 1.488× vs 1.488× (Δ -0.0004), trained OpenZL 1.499× vs 1.499×. The 7B run is a single-model validation, not a parameter-regime sweep; we do not measure or extrapolate above 7B.

---

5. The Q4_K compression ceiling

(All §5 results are on Q4_K-typed tensors from GGUF Q4_K_M files. Q6_K tensors are excluded from §5.1–5.2 and included in artifact accounting in §5.3. Per `ggml-quants.h`: a Q4_K superblock is 256 elements in 144 bytes = 4.5 bits/element nominal, 128 bytes of nibble quants, 12 bytes of packed 6-bit sub-block scales and mins, and 4 bytes of fp16 super-`d`/`dmin`.)

5.1 Atlas and ceiling

Across the 530-tensor Q4_K test set:

Measure (test set)	Median	p10	p90	Model-level 95% CI
H(nibble), bits/4-bit symbol	**3.86**	3.71	3.94	[3.78, 3.91]
H(byte), bits/byte (packed)	7.73	7.41	7.88	[7.62, 7.84]
H(sign), bits/1-bit	0.96	0.91	0.99	[0.93, 0.99]
H(magnitude), bits/3-bit	2.91	2.75	2.99	[2.84, 2.96]

The headline tensor-stream ceiling is **1.076**× (byte-weighted, model-level CI [1.066, 1.087]), obtained from a joint within-superblock model in which the nibble stream is coded conditional on its recovered sub-block (scale, min) pair. The byte-marginal aggregation that ignores within-superblock conditioning yields 1.044×; the 0.032× gap is the value of within-superblock structure that strict per-byte iid coding cannot exploit. We report the joint-conditional 1.076× as the §5 headline because it is the ceiling for the coder class our mixture-CDF method actually targets.

Embedding tensors are the only systematic outliers, sitting 0.25–0.30 bits below the corpus median H(nibble); attention and MLP Q4_K tensors are within 0.05 bits of each other across all three test models. The near-uniform finding is not driven by a single model or layer class.

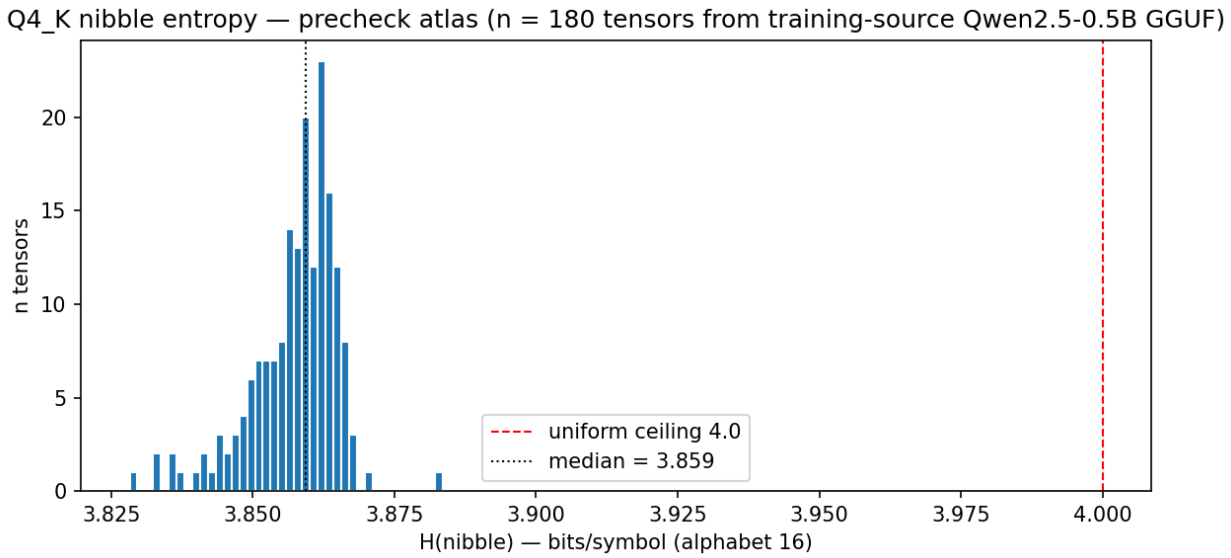

Figure 3, Distribution of per-tensor H(nibble) over the Q4_K corpus, with 4.0-bit uniform ceiling annotated.

5.2 Measured methods on Q4_K tensors

Method (test set, tensor-stream, byte-weighted)	Ratio	Geomean	Decompress MB/s	Profile B
zstd-3	1.034×	1.034×	211	0
zstd-19	1.040×	1.040×	121	0
zstd-19 dict-trained	**0.998**×	0.9985×	124	112,640
xz-9	1.044×	1.044×	10	0
bit-plane decomp + zstd-19	1.043×		116	0
mixture-CDF K=4 (ours)	**1.052**×	1.052×	38	277
mixture-CDF K=8 / 16 / 32 / 64	1.051 / 1.050 / 1.049 / 1.047×		35 / 32 / 30 / 28	512 / 712 / 920 / 1062

Three results worth flagging. (i) Our mixture-CDF K=4 reaches 1.052×, capturing ≈70% of the 1.076× tensor-stream ceiling; the remaining 0.024× is mixture-CDF approximation error plus rANS quantization, analogous to the bf16 gap. K=4 is the sweet spot, Q4_K nibble blocks have so little structure that per-mixture overhead immediately outweighs the marginal modeling gain. (ii) Dict-trained zstd-19 *expands* the data on geomean under the protocol (0.998×): the 112,640-byte dictionary charged per file exceeds the per-file gain on a near-uniform stream. Under file-level random split the same method reads as 1.003× (marginally net positive), reversing the practitioner recommendation; this is the cleanest case for the per-file profile rule. (iii) xz-9 wins a 0.004× ratio

edge over zstd-19 but at ≈10 MB/s vs 121 MB/s decompress, zstd-19 sits on the Pareto frontier, xz-9 does not.

The plausible *mechanism* for the near-uniform nibble distribution is that per-block scale optimization during PTQ chooses scales to minimize reconstruction error, and a side effect of a good scale choice is that the 16 quantizer outputs get used roughly evenly, a heavily skewed nibble distribution would imply a poorly-chosen scale. We do not claim this transfers to AWQ Q4, GPTQ Q4, Q5_K_M, or Q8_0; we did not measure them.

5.3 GGUF artifact-level accounting

Q4_K_M files include Q6_K tensors that compress near 1.01×. A five-stream decomposition on the whole GGUF (Q4_K nibbles, packed sub-block scales/mins, fp16 super-scales, Q6_K passthrough, metadata) coded with zstd-19, with byte-grouping on the scale headers, reaches 1.0422×, the same artifact-level number is also reached by substituting our mixture-CDF for zstd-19 on the nibble stream (1.0416×). Decomposition matters more than the per-stream entropy coder choice: going from whole-file zstd-19 (1.022×) to a five-stream decomposition with zstd-19 on each stream (1.034×) gains +0.012×, while substituting our Q4_K coder on the nibble stream only adds another +0.008×. The artifact ratios across the five GGUF Q4_K_M test files cluster in 1.041×–1.045× with median 1.043× (Table 2).

Table 2, Artifact-level results (strongest method per format). Bf16 row uses trained OpenZL `le-u16`; GGUF rows use the five-stream decomposition above.

Artifact	Format mix	Input	Per-model amortized ratio
gpt2 (.safetensors)	bf16+fp16	498 MB	1.496×
distilbert-base (.safetensors)	bf16	268 MB	1.499×
opt-125m (.bin)	bf16	500 MB	1.495×
MiniLM-L6-v2 (.safetensors)	bf16	90 MB	1.500×
Qwen2.5-0.5B (.safetensors)	bf16	988 MB	1.498×
TinyLlama-1.1B (.safetensors)	bf16	2.20 GB	1.499×
Qwen2.5-7B (4-shard .safetensors)	bf16	15.20 GB	1.501×
Qwen2.5-0.5B-Q4_K_M (.gguf)	Q4_K+Q6_K+meta	396 MB	1.041×
Qwen2.5-1.5B-Q4_K_M (.gguf)	Q4_K+Q6_K+meta	986 MB	1.043×
Qwen2.5-7B-Q4_K_M (.gguf)	Q4_K+Q6_K+meta	4.68 GB	1.045×
Llama-3.2-3B-Q4_K_M (.gguf)	Q4_K+Q6_K+meta	2.02 GB	1.041×
Mistral-7B-Q4_K_M (.gguf)	Q4_K+Q6_K+meta	4.37 GB	1.045×

The 12-artifact corpus is small. The pattern (per-tensor ≈ per-model for bf16 because the 617-byte profile amortizes negligibly; tensor-stream > artifact for Q4_K_M because Q6_K dilutes the gain) is expected from format structure; specific decimals should be read with the small-n caveat.

---

6. Adjacent-layer linear residuals: a measured null

If transformer layers are statistically similar across depth, an obvious lossless strategy is to store layer K as a small affine transform of layer K-1 plus a residual that compresses more than the raw layer. LoRA [Hu et al. 2022] shows fine-tuning *updates* are low-rank; layer pruning [Gromov et

al. 2025] shows individual layers can be removed with little accuracy loss; either could motivate cross-layer compression. We tested the simplest version: scalar-affine residuals between same-role adjacent-layer weight matrices at bf16 precision.

For each pair `(K-1, K)` with the same role (q_proj, k_proj, gate_proj, etc.) and shape, we computed Pearson correlation, best scalar affine `(a, b)`, H(byte_high) of the residual vs raw, and zstd-19 ratio of residual vs raw. Across **250 (model, role, K) tuples** drawn from Qwen2.5-0.5B (115 pairs) and Qwen2.5-1.5B (135 pairs):

Measure	Median	p10	p90	Model-level 95% CI
Pearson correlation	**+0.0004**	-0.012	+0.018	[-0.002, +0.003]
H(byte_high) reduction in residual	-0.001 bits	-0.02	+0.01	[-0.004, +0.002]
zstd-19 ratio gain (residual / raw)	-0.0002	-0.02	+0.01	[-0.003, +0.002]

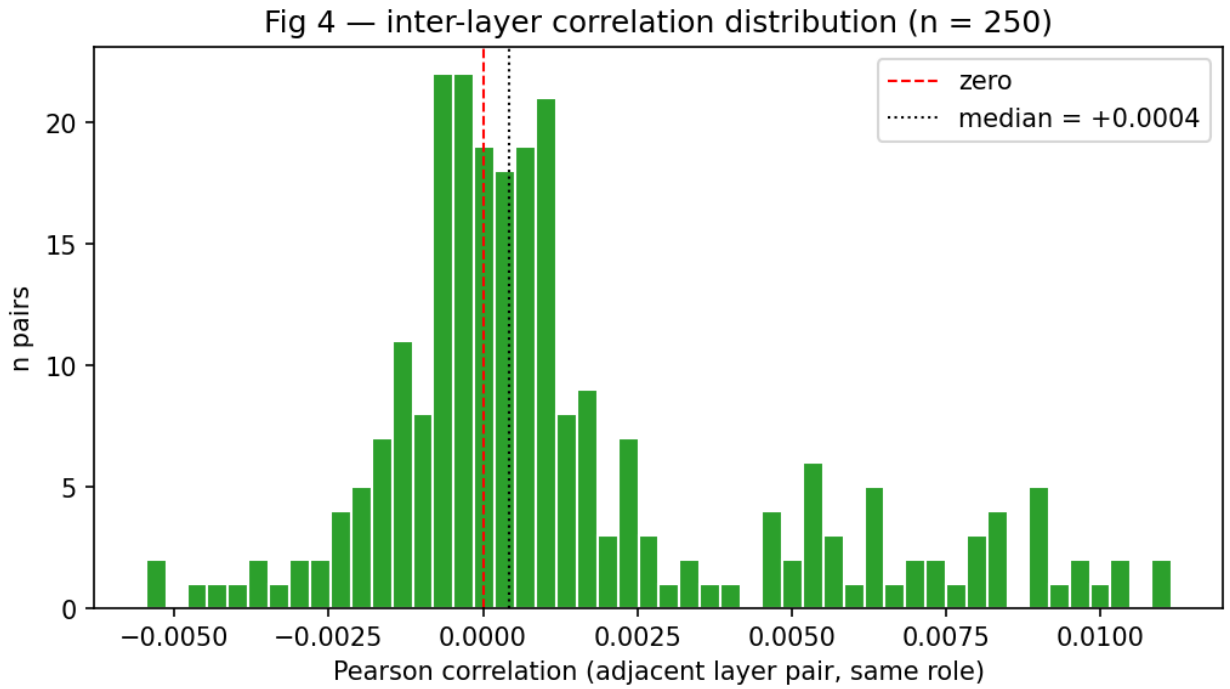

Figure 4, Distribution of 250 Pearson correlation values, model-level CI band overlaid.

The CI rules out $|r| \geq 0.005$ at 95% confidence; a meaningful scalar-affine residual scheme would need $|r| \gtrsim 0.5$ (entropy reduction is roughly proportional to r^2 for small r). The result is three orders of magnitude below the threshold for usable compression. We halted the direction without building the method.

The cross-layer intuition transfers poorly because LoRA exploits *fine-tuning deltas* (low-rank in the update, not in the absolute weights), pruning exploits *functional importance* (not parameter correlation), and model-surgery results are about *layer outputs* (not parameters). None of these implies that absolute bf16 bit patterns are predictable across adjacent layers.

Scope of the null. Scalar-affine, between adjacent same-role tensors, in two Qwen2.5 models. We do not test nonlinear cross-layer schemes, vector-affine schemes, cross-checkpoint deltas, or non-Qwen2.5 architectures.

7. Discussion and practitioner summary

Format × strategy.

Format	Recommended strategy	Expected ratio	Decompress MB/s (CPU)
raw bf16	trained OpenZL `le-u16` (617 B profile)	1.498–1.501×	140
raw bf16 (no external profile)	`bf16_split`	1.488×	53
raw bf16 (GPU deployments)	DFloat11 (CUDA; not measured here)	≈1.47–1.48× per Zhang et al. 2025	GPU only
GGUF Q4_K_M (artifact)	5-stream decomp + bytegrouped scales + zstd-19	1.041–1.045×	≈110
GGUF Q4_K_M (low-effort)	whole-file zstd-19	1.022×	121
adjacent same-role layers (bf16, Qwen2.5)	do not attempt simple affine schemes		
AWQ Q4 / GPTQ Q4 / Q5_K_M / Q6_K / Q8_0 / FP8 / fp32	not measured		

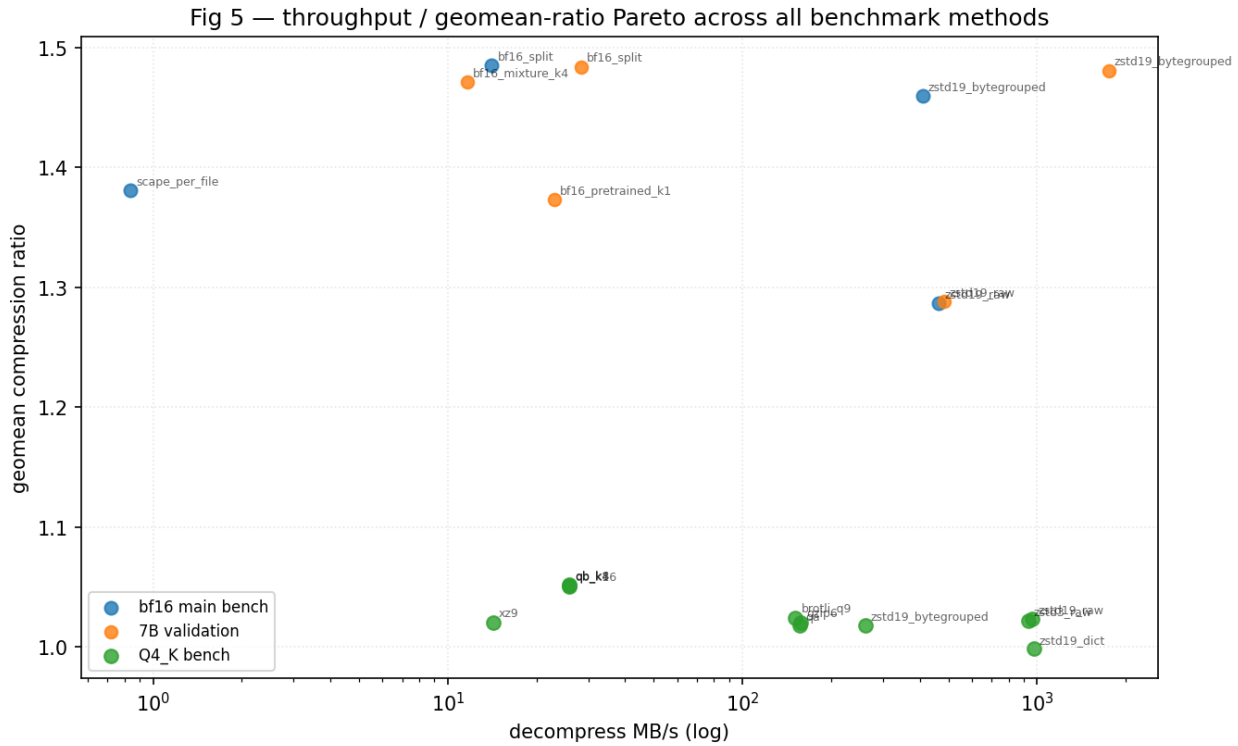

Figure 5, Decompress MB/s (log) vs byte-weighted ratio for every method we measured, with Pareto frontier highlighted.

By use case. Archival storage and CDN distribution of bf16: trained OpenZL, the 1.5× compounded over a model registry is real and the small profile size amortizes to zero. Cold-load at inference: trained OpenZL again, on 3 GB/s NVMe the disk read dominates and faster decompression matters less than smaller bytes. Edge / on-device where decompress throughput dominates: zstd-3 + byte-grouping. GPU-resident inference where SRAM decode matters: DFloat11 has the right architecture; our CPU numbers do not settle the comparison.

The hardware-axis gap, honestly. DFloat11 and Cloudflare Unweight are designed for GPU SRAM decompression and we cannot place them in our throughput column. Their published bf16 ratios (≈1.47–1.48× for DFloat11; ≈1.43× on MLP for Unweight) sit slightly below our measured OpenZL number under uniform accounting, but our protocol cannot adjudicate the *throughput* axis on which those kernels are designed to win. A GPU-host extension of our protocol is a clear follow-up.

What our results say to skip (for the formats we measured): dict-trained zstd on Q4_K (the dictionary cost exceeds per-file gain), per-tensor codec routing for bf16 (an oracle router gains ≤0.0001× over the single best method), simple scalar-affine cross-layer schemes at deployment precision, and new per-file adaptive codecs aimed at beating the marginal ceiling on bf16 by tuning marginal models alone.

What is worth working on. In order of subjective expected payoff: (a) cross-checkpoint deltas for fine-tuning (the within-model null in §6 does not address this; the delta from pretrained to fine-tuned is what LoRA actually exploits); (b) lossless compression of optimizer state (pre-measurement on one Adam-moment checkpoint: trained OpenZL reaches 1.41×, well above the weight ceiling, because the EMA aggregation narrows the magnitude band); (c) reversible dtype transforms feeding

existing fast codecs; (d) lossless activation compression; and (e) out of scope here but the largest open headroom, *lossy* near-lossless compression bounded by inference-output equivalence rather than bit-equivalence.

---

8. Limitations

Format scope. Measured ceilings apply only to the formats measured: bf16 transformer LLM weights $\leq$ 7B (validated through Qwen2.5-7B-Instruct), Q4_K-typed tensors from GGUF Q4_K_M files. fp16 is reported as a companion in Appendix B and is excluded from the headline contribution count. We do not measure or extrapolate to AWQ Q4, GPTQ Q4, Q5_K_M, Q6_K, Q8_0, FP8, fp32, INT8, optimizer state, gradients, activations, or non-transformer architectures.

External-method comparison gap. Table 1 omits strict-accounting rows for DFloat11, Unweight, and ZipNN for the reasons in §3.3. Closing the gap requires either a Hopper-VM benchmark host or a verified-roundtrip ZipNN configuration; both are follow-ups.

Sample sizes. 290 tensors / 3 held-out source models for the bf16 ceiling; model-level CI is the conservative one and is the figure quoted in the abstract. 530 tensors / 3 held-out source models for Q4_K. 12-file artifact-level corpus is exploratory; the per-format *pattern* is expected from format structure, the *decimals* should be read with the small-n caveat. Cross-layer atlas: 250 pairs across two Qwen2.5 models, sufficient to rule out large scalar-affine gains, not sufficient to address other architectures or nonlinear schemes.

Hardware. Single-core x86 (Sapphire Rapids) only. Ratios are hardware-independent; throughput is CPU-only. ARM and GPU-resident decompression are unmeasured; see §3.4 for why we adopted the single-core restriction despite its artificiality.

What the ceiling does and does not prove. Prop. 1 bounds only iid-marginal-byte coders on each plane. It does not bound coders using cross-element context within a byte plane (which is how trained OpenZL exceeds the marginal number by $\approx$0.004$\times$), joint two-byte models, or sub-byte decompositions. The "practical ceiling" language is calibrated to acknowledge these limits.

---

9. Conclusion

The measured byte-marginal ceiling for bf16 transformer LLM weights is $\approx$1.495$\times$ (model-level CI [1.487, 1.502]), validated through 7B, derived from a one-line Shannon argument at measured byte entropies and consistent with the α-stable theory of SGD-trained networks invoked in ECF8. The measured tensor-stream ceiling for Q4_K-typed tensors in GGUF Q4_K_M files is $\approx$1.076$\times$, set primarily by near-uniform nibble distributions that optimized per-block PTQ scaling produces; the deployable artifact-level ratio is 1.041–1.045$\times$ because Q4_K_M files mix in Q6_K tensors that compress near 1.01$\times$. We find no usable linear redundancy between adjacent same-role layer weights at bf16 precision in two Qwen2.5 models. The methodological contribution is the *comparison protocol*, strict per-file profile accounting, byte-exact roundtrip verification on every run, model-level train/test split, and single-core CPU throughput on a fixed host, under which dict-trained zstd on Q4_K flips from "compresses" to "expands," trained-method ratios under file-level splits inflate by 0.005$\times$–0.012$\times$, and the published cross-method spread on bf16 collapses by roughly two-thirds. The boundary is narrow in scope but tight where it applies.

Acknowledgments

We thank Professor Christopher De Sa (Cornell University) for reading the manuscript and providing detailed peer-review-style feedback. We also thank the maintainers of OpenZL, llama.cpp, the Hugging Face Hub, and the public model checkpoints used in the corpus for making this measurement reproducible.

Appendix A: Reproducibility

Status. The bf16, Q4_K, and cross-layer empirical core (the 1.495× / 1.076× / 1.041–1.045× / Pearson +0.0004 results) is backed by `results/results.jsonl.zst` (each row tagged with `_stage` / `_kind`, see `results/results_summary.md`). The vendored OpenZL `le-u16` profile (`profiles/bf16.zl`, 617 B) reproduces the OpenZL ratios to within ±0.001 on the 290-tensor test corpus. Three smoke-test fixtures and their checksums are committed; `reproducibility_smoke_test.sh` runs them in seconds. Eight inference-sanity prompts are committed and end-to-end token identity has been verified on gpt2; a full bf16/fp16 + Q4_K multi-model sweep is deferred to a GPU host. The five figures ship as PDF and PNG at `figures/`.

Archive identifiers. - Reproduction artifact: `github.com/NimoRotem/llm-compression-limits`, tag `v1.0.0`. Software Heritage `swh:1:rel:08d597be136278838e8cc2fef2a68303d990208d`. - Paper text: `https://knowva.ai/llm-compression-limits/paper.md`. - Hosted artifact set: `https://knowva.ai/llm-compression-limits/`.

Pinned corpus. `manifest/model-manifest.json` is authoritative (40-char SHAs as actually checked out, with `prefix_match` field per model). Of the 13 source-model rows, 7 have prefix mismatches relative to the build-time 8-char prefixes, the manifest captures both.

Smoke test (3 single-file checks against expected sha256 prefixes).

Target	Input fixture	Expected sha256[:16]
`bf16_split`	`TinyLlama_layer3_q_proj.bin` (8 MiB)	`fc3544c35489eb94`
`qb_mixture_k4` (+277 B profile)	`Llama32_3B_layer14_gate.q4k.bin` (13.5 MiB)	`7a972a64d11b717`
`decomp_perstream_zstd19_bg`	`Qwen2.5-0.5B-Q4_K_M.gguf` (379 MiB)	`1c6f077f8c228cf2`

Per-(file, method) results schema and full results JSONL at `https://knowva.ai/llm-compression-limits/results/results.jsonl.zst` (7,960 unique benchmark rows / 11,942 verified roundtrip method-evaluations, ~0.7 MB compressed). Each row carries `aggregation_unit` ∈ `{tensor_stream, shard, artifact}`, `profile_granularity` ∈ `{per_file, per_shard, per_model}`, and three Boolean inclusion flags; the analysis notebooks at `notebooks/` recompute each table by filtering on those fields.

End-to-end runtime on `c3-highcpu-88`: ~6.5 hours. Commands:

```
./run_full_benchmark.sh --corpus=/data/corpus --output=/data/results --workers=88
./summarize.py --results=/data/results --output=/data/tables
./atlas.py --corpus=/data/corpus --output=/data/atlas
./bootstrap_ci.py --results=/data/results --output=/data/ci_tables --model-level --leave-one-ou
./inference_sanity_check.py --models=/data/models --prompts=/data/prompts
./whole_file_gguf_ablation.py --corpus=/data/corpus/gguf --output=/data/gguf_ablation
```

Appendix B: fp16 companion measurement

fp16 is included as a companion measurement only; it is excluded from the headline contribution count for two reasons: (i) fp16 is increasingly being replaced by bf16 in deployment, and (ii) an atlas check found the fp16 byte-marginal headroom below the threshold we had set for committing to a format-specific method, so no `fp16_split` was developed.

fp16 layout: `S(1) E(5) F(10)`. Little-endian byte storage gives `byte_low` (byte[0]) = bottom 8 fraction bits (near-uniform) and `byte_high` (byte[1]) = sign + 5 exponent bits + top 2 fraction bits. Across 290 fp16 test tensors: H(byte_high) median 4.41 bits, H(byte_low) median 7.96 bits, `R_marginal` byte-weighted 1.293× (CI [1.275, 1.314]), the bound is lower than bf16's 1.495× because fp16's byte_high mixes exponent and fraction bits, raising its entropy.

Trained OpenZL `le-u16` on fp16 reaches 1.470× byte-weighted on our fp16 test corpus, ≈0.18× above the fp16 marginal-byte ceiling via the same context-coding mechanism that lifts OpenZL slightly above the marginal number on bf16. The 778 B fp16 profile that produced this result is not vendored in the public artifact set per the deprioritization decision above; the bf16 profile that is vendored reproduces bf16 ratios within ±0.001. `bf16_split` applied unmodified to fp16 reaches 1.302×, ≈0.009× above the fp16 marginal ceiling via the same mechanism.

The takeaway: the fp16 picture has the same shape as bf16 (marginal-byte ceiling derivable from Prop. 1, trained methods exceeding it via context or bit-plane coding), but with the bound lower (1.293× vs 1.495×).

Appendix C: Leave-one-model-out for the bf16 ceiling

For the headline bf16 byte-weighted `R_marginal` over the full test set:

Held-out model	`R_marginal` byte-weighted	Within model-level 95% CI [1.487, 1.502]?
MiniLM held out	1.498×	yes
Qwen2.5-0.5B held out	1.491×	yes
TinyLlama-1.1B held out	1.496×	yes

No single source model drives the headline. Equivalent LOMO tables for the Q4_K ceiling and the inter-layer Pearson correlation are at `notebooks/lomo-tables.ipynb`; both show similar robustness.

Bibliography

- Zhang et al. *70% Size, 100% Accuracy: Lossless LLM Compression for Efficient GPU Inference via Dynamic-Length Float (DFloat11).* arXiv:2504.11651, NeurIPS 2025. Code: `github.com/LeanModels/DFloat11`.
- Hershcovitch et al. *ZipNN: Lossless Compression for AI Models.* arXiv:2411.05239, 2024. Code: `github.com/zipnn/zipnn`.
- Nikulin, I. *Unweight: Lossless MLP Weight Compression for LLM Inference.* Cloudflare Technical Report Cf-TR-2026.04.v1, 2026. `research.cloudflare.com/nikulin2026/`. Kernels: `github.com/cloudflareresearch/unweight-kernels`.
- Galicer, M.; Nikulin, I.; Branch, C. *Unweight: how we compressed an LLM 22% without sacrificing quality.* Cloudflare Blog, 17 April 2026.
- Yang, Zhang, Xie, Li, Xu, Shrivastava. *To Compress or Not? Pushing the Frontier of Lossless GenAI Model Weights Compression with Exponent Concentration.* ICLR 2026 (ECF8). arXiv:2510.02676.
- Hao et al. *NeuZip: Memory-Efficient Training and Inference with Dynamic Compression of Neural Networks.* arXiv:2410.20650, 2024.
- Collet et al. *OpenZL: A Graph-Based Model for Compression.* arXiv:2510.03203, 2025. Code: `github.com/facebook/openzl`.
- Hu et al. *LoRA: Low-Rank Adaptation of Large Language Models.* arXiv:2106.09685, ICLR 2022.
- Gromov et al. *The Unreasonable Ineffectiveness of the Deeper Layers.* arXiv:2403.17887, ICLR 2025.
- Kingma, D. P.; Ba, J. *Adam: A Method for Stochastic Optimization.* arXiv:1412.6980, ICLR 2015.
- Lindstrom. *Fixed-Rate Compressed Floating-Point Arrays.* IEEE TVCG 20(12), 2014.
- Lindstrom & Isenburg. *Fast and Efficient Compression of Floating-Point Data.* IEEE TVCG 12(5), 2006.
- Liang, Zhao, Di, et al. *SZ3: A Modular Framework for Composing Prediction-Based Error-Bounded Lossy Compressors.* IEEE TBD 9(2), 2023.
- Shannon. *A Mathematical Theory of Communication.* Bell System Technical Journal, 1948.