Below One Bit, Small Models Quantized Without Retraining Fall Past the Byte-Entropy Floor

Anonymous Author(s)

Abstract

Below one bit per weight, quantizing a trained model does not merely degrade it: it carries the model past the entropy floor of the data. In small byte-level transformers we compare native quantization-aware training (QAT) from a random initialization against three post-hoc routes at matched, honestly counted bit-width: a strong calibration post-training quantization (PTQ) baseline, a low-rank binary factorization, and LB-ADMM, the reference sub-1-bit solver. At 1.09 counted effective bits per weight, all three stay below the corpus's order-0 byte-frequency floor of 5.06 bits per byte (bpb). At the two sub-1-bit widths we test, 0.59 and 0.41, all three land at or past it: the compressed model predicts the next byte worse than a table of marginal byte frequencies, while native QAT holds near 3.1. The floor belongs to the corpus, not to any method, so its height is not the result; where the routes cross it is, and that crossing falls near one bit. The gap runs 1.9 to 3.4 bpb across the three post-hoc classes at the anchor and does not narrow across a token-matched sweep from 1.26M to 19.18M parameters, where it holds at 2.2 to 4.4 bpb against the calibration baseline. What closes it is a step the compression literature identified above one bit: training-scale optimization against the deployed representation, which keeps working below the floor from each starting route we test. Retraining only the 8.4% full-precision interface, leaving every quantized weight bit-frozen as PTQ shipped it, recovers most of the collapse. Repair is cheap in parameters and expensive in tokens. This is a small-scale result, not a claim about billion-parameter LLMs.

1 Introduction

This paper asks a simple question: when a model is compressed below one bit per weight, how much of the loss comes from the representation, and how much from projecting a trained model into it? We separate the two with a controlled, small-scale comparison, and the projection step turns out to carry almost all of it. The result worth reporting is not that the two differ but how far apart they land. Below one bit, every post-hoc route we test falls past the order-0 byte-frequency floor of the corpus: the compressed model predicts the next byte worse than a table of marginal byte frequencies does. A model trained directly at the identical width does not. The floor's height is a fact about enwik8; where the routes cross it, near one bit rather than anywhere along the compression ladder, is the finding.

Why the projection step costs so much is a question the compression literature has largely answered above one bit. Pruning work established that a surviving support is worth little until its values are resolved by retraining [Renda et al., 2020, Liu et al., 2022a, Wagner et al., 2025], low-bit repair work showed that adjusting only the full-precision interface around quantized weights recovers much of the loss [Li et al., 2023], and recent analysis explains the mechanism: gradients taken at quantized weights bias toward the low-loss basin of the deployed representation [Li et al., 2026]. Our contribution is to carry that account into a regime none of this work tests. Below one bit the same condition, optimization at training scale against the representation that will ship, still governs, and it remains order-agnostic: it is met by a from-scratch run, by a fine-tune started from the post-hoc solution,

and by an adaptation that never touches a quantized value at all. Order-agnostic means every one of these routes closes most or all of the gap, not that they cost the same: which is cheapest is a compute-allocation question, studied above one bit by Dremov et al. [2026] and left open below it.

Sub-1-bit weight quantization of large language models starts, almost without exception, from a model that is already trained (Section 4), and the methods we survey are demonstrated at billion scale, where parameter redundancy is abundant. That entangles the two explanations at the scale where sub-1-bit quantization is normally studied. At small scale redundancy is scarce, so a controlled comparison can isolate the cost of the projection itself. The alternative to rounding is to train directly for the target precision: native quantization-aware training (QAT) from a random initialization, well studied at and above one bit [Wang et al., 2023, Ma et al., 2024] but rarely used below one bit even as a comparison point, on cost grounds that do not hold at the sub-million-parameter scale, where a full sweep runs in minutes on a laptop. We run the comparison directly, against three post-hoc baselines. Our contributions are:

- A controlled comparison at matched counted bit-width, at under one million parameters: strong calibration PTQ (GPTQ-style error feedback plus Hessian-optimal per-row scaling over $n{:}m$ masks), binary factorization, and Chong et al. [2026]'s own LB-ADMM solver all fall 1.9 to 3.4 bits per byte short of native QAT, separating completely in every seed (Section 3.1). All three clear the corpus's order-0 byte-frequency floor at 1.09 bits (5.00, 5.03 and 4.80 bpb against 5.06), though only LB-ADMM clears it by more than one standard deviation, and none of them clears it at either sub-1-bit width.
- A two-budget fine-tuning experiment showing the post-hoc deficit is a budget effect, not a floor: a QAT fine-tune of the PTQ weights at 20% of the original tokens leaves a 0.3 bpb deficit below one bit, while the full original budget closes it at every width, ending 0.09 to 0.38 bpb below from-scratch QAT at twice the total token cost (Section 3.2).
- A decomposition locating the gap in the missing re-solve rather than the representation: with values re-solved against the deployed weights, neither support content (a random support nearly matches the calibrated one) nor initialization is resolved from zero at our measured run-to-run floor of 0.05 bpb, and the calibration estimate converges without improving, while retraining only the full-precision interface (8.4% of parameters) around bit-frozen weights recovers 97% of the collapse at the anchor (Section 3.3) and 89% at a second, wider configuration (Appendix K). Adapting the interface is most of the repair, not all of it, and it is cheap in parameters but not in tokens (Appendix E).
- A token-matched scale sweep fixing tokens per parameter across a fifteenfold range (1.26M to 19.18M parameters): the gap does not narrow, and holding tokens per parameter constant removes the undertraining confound of a fixed token budget (Section 3.4).

Scope and limitations, up front. These are small-scale, byte-level experiments: a controlled separation result, not a general claim about post-hoc quantization or about billion-parameter LLMs. Section 4 expands each limitation.

2 Method

Model and data. We train a small GPT-style transformer from a random initialization on byte-level **enwik8** [Mahoney, 2011], the classic character-level compression benchmark whose native metric is bits per byte (a random byte is 8.0 bits per byte). We choose byte-level modeling because it avoids a learned tokenizer embedding that would absorb part of the quantization burden: here every weight-flop is on the quantized block linears, and the bpb metric carries an absolute reference point. We compare throughout against the corpus's order-0 byte-frequency floor, or byte-entropy floor: the entropy $H_0 = -\sum_b p(b) \log_2 p(b)$ of the validation split's marginal byte distribution (5.06 bpb on enwik8), the score of a model that predicts byte frequencies and nothing else.

Quantization-aware training. Only the transformer-block linear layers are quantized. A straight-through estimator [Bengio et al., 2013] trains the underlying full-precision weights while the forward pass uses the quantized weights: $\hat{w} = w + \mathrm{sg}(q(w) - w)$, where sg is stop-gradient and q the quantizer. We study three families: `fp16`, the full-precision reference, which names a storage budget of 16 bits per weight rather than an arithmetic format (every arm, this one included, computes in

float32; only the counted bit-width differs); `binary`, one-bit $\mathrm{sign}(w) \cdot s_{\text{row}}$ with a per-row scale in the style of Wang et al. [2023] (1.09 effective bits per weight after scale overhead); and `subbit` $n{:}m$, structured-sparse binarization that, within each contiguous block of m weights, keeps the n largest-magnitude survivors, binarizes them, and applies a per-row scale, with the rest set to zero. The last family is genuinely below one bit.

Transparent effective bit-width. Sub-1-bit comparisons depend strongly on what is included in the bit count. We count, per weight, the survivor sign bits (n/m), the $n{:}m$ index that records *which* weights survive ($\log_2 \binom{m}{n}/m$), and the amortized full-precision per-row scale ($16/\text{in_features}$). The index is the dominant term, and including it is what makes these genuine sub-1-bit numbers rather than an estimate that assumes a free mask. Under this accounting, 1:8 is 0.59 and 1:16 is 0.41 effective bits per weight at the anchor width (per-component breakdown in Table 3, Appendix A).
Scope of the count. Every reported effective bpw covers the quantized transformer-block linear weights only. The embeddings, layer norms, and output head remain full precision in every arm and are excluded from the count, as is standard practice [Wang et al., 2023, Ma et al., 2024]; they are held identical across arms, so the exclusion favors neither. Section 3.3 shows this full-precision interface is nonetheless load-bearing for repair.

The three post-hoc baselines. Each quantizes the block linears of an already-trained full-precision model at the same counted bit-width the native arm trains into.

(i) Strong calibration PTQ. A real GPTQ routine [Frantar et al., 2022]: per-layer Hessian $\mathbb{E}[xx^\top]$ from 200k calibration tokens, Cholesky-stabilized inverse, per-output-channel asymmetric k-bit uniform with column-sequential error feedback. The $n{:}m$ arms add a Hessian-optimal per-row scale, the least-squares-optimal scalar for the row's survivor pattern under the calibration Hessian, and take the best mask per row over {magnitude, salient/AWQ (activation-aware weight quantization) [Lin et al., 2024], round-to-nearest (RTN)} by calibration Hessian error (Appendix A). RTN is always a candidate, so the strong variant is at least as good as RTN *on the calibration objective* by construction. It is not thereby better on the loss, and below one bit it is not: that gap between the objective and the loss is the subject of Section 3.3. Two gates clear the machinery before any sub-1-bit number is trusted: 4-bit uniform GPTQ is near-lossless against fp16 (2.476 vs 2.449 bpb, six seeds; the full uniform ladder is in Appendix G), and the strong $n{:}m$ variant beats naive RTN at binary ($+0.38$ bpb, in all six seeds). Below one bit that margin over RTN stops being a reliable win, which is itself a symptom of the objective-loss decoupling rather than a defect of the baseline (Appendix G).

(ii) Binary factorization, solved post-hoc [Lee et al., 2025]. A low-rank factorization $W \approx U_b \mathrm{diag}(s) V_b^\top$ with binary factors $U_b, V_b \in \{\pm 1\}$ and least-squares scales, SVD-initialized and refined by binary coordinate descent. One disclosure matters here, for the same reason it does in (iii): LittleBit does not solve this representation post-hoc. It *trains* the factors, using its Dual-SVID decomposition as a quantization-aware training initialization and learning its compensation terms by gradient descent with distillation. We do not reproduce that pipeline and do not claim to. What we run is its *representation*, reached by projection instead of by training: the same container, entered the other way. Rank is matched to the bit budget by the same transparent accounting rather than hand-tuned. Allowing full rank tests the representation's own ceiling, which still sits $+2.17$ bpb above fp16 ($n{=}1$), so no rank budget recovers fp16 here (Appendix A).

(iii) LB-ADMM [Chong et al., 2026]. A from-scratch baseline invites the objection that the solver, not the class, is what failed, so we also run the reference solver itself, in a port that agrees with its authors' implementation to $1\mathrm{e}{-7}$. It gets every advantage we can give it: its own default iteration budget, the calibration statistics of the calibration-PTQ arm, so a difference between them is the algorithm and not the data, and each seed's own model, so it is paired by seed like the other arms. Its rank follows the same accounting, $r(\text{out} + \text{in})$ sign bits plus $16(\text{out} + \text{in} + 2r)$ scale bits, and lands on the budget rather than near it (0.401 against a 0.406 target at 1:16).

Fair learning rate. A single shared learning rate is unfair across bit-widths: full precision prefers a lower rate than straight-through-trained models (a shared rate once made a one-bit model spuriously beat fp16). Every arm is therefore tuned inside its own protocol and never at a rate borrowed from another, since optima do not transfer across bit-width, architecture, or adaptation protocol. Two selection procedures appear in this paper and we name which is in force wherever a rate is reported. The anchor arms select by lowest mean over three seeds; every other cell, including the scale sweep,

the further corpora, and each fine-tune and repair arm, brackets at seed 0 and is reported only at a rate whose two grid neighbors are both worse. Grids, selected rates, and the arms whose winner sits at the edge of its grid (a bound, not a confirmed optimum) are listed in Appendix C.

Reproducibility. All experiments run on a single Apple Silicon laptop under MLX. An anchor-scale cell trains in one to three minutes, which is what makes a per-arm bracketed rate affordable and lets every reported cell be a from-scratch rerun rather than an estimate. Learning-rate selection reuses seeds that also appear in the evaluation set, which would leave the reported means mildly optimistic, so we re-select on disjoint seeds as a check: the baseline's rate is confirmed and no reported mean moves (Appendix C). Seed counts are given in each caption; the optimizer, schedule, hardware, corpora, splits, and licenses are in Appendix B.

3 Results

Setup. The headline is anchored at a model trained to a fixed budget of about 29 tokens per parameter (embedding width 128, four layers, four attention heads, a $4\times$ feed-forward width, context 256, batch 64; 0.86M parameters, 25M training tokens, about 29 tokens per parameter). All numbers are mean $\pm$ standard deviation over six independent runs; the metric is validation bits per byte (lower is better). Seed spread is widest for full-precision fp16 (± 0.238, small-model instability at full precision), intermediate for `binary` (± 0.141), and tight for the sub-1-bit arms (± 0.046 at 1:8, ± 0.060 at 1:16).

One statistical convention, stated here and used throughout. A gap quoted in pooled standard deviations is an effect size, not a significance level: at six seeds a two-digit standardized effect is not a number to read literally. The arms are paired by seed, seed i fixing the same initialization and data order in each, so the evidence we lean on is the sign of the paired difference. An exact one-sided sign test on six paired differences gives $p = 2^{-6} = 0.016$, the smallest value six pairs can produce. We attach no per-cell p anywhere and rest the argument on the size of each gap and on the completeness of the separation.

3.1 The direct contrast: training beats rounding below one bit

Holding model, data, and metric fixed, we compare two ways to reach the same sub-1-bit representation. The **native QAT** arm trains from a random initialization with the quantizer in the forward pass: same architecture, data, and seed as full precision, and no warm start from the trained weights. The **post-hoc** arms quantize the already-trained full-precision weights after the fact, all three applied to the block linears of the identical trained model, so a difference between the two paths is the path and not the model (Section 2; the two routes are diagrammed in Figure 2, Appendix A).

Table 1: Native QAT versus three evaluated post-hoc classes at matched effective bit-width (enwik8, 0.86M anchor; every arm is 6 runs; mean $\pm$ std bits per byte, each arm at its own bracketed best learning rate; gap = calibration PTQ minus native QAT in bits per byte, with the pooled-standard-deviation equivalent in parentheses; a random byte is 8.0 bits per byte). LB-ADMM is the reference sub-1-bit post-hoc solver of Chong et al. [2026], run at its own default iteration budget on the same calibration statistics and the same per-seed models.

		tested post-hoc PTQ			
eff. bpw	native QAT	calibration	factorization	LB-ADMM	gap (calib.)
0.41 (1:16)	3.073 ± 0.060	6.447 ± 0.551	6.308 ± 0.342	6.470 ± 0.739	$+3.37$ (9σ)
0.59 (1:8)	2.921 ± 0.046	5.806 ± 0.204	5.670 ± 0.301	5.845 ± 0.429	$+2.89$ (19σ)
1.09 (binary)	2.879 ± 0.141	5.004 ± 0.316	5.033 ± 0.236	4.800 ± 0.206	$+2.12$ (9σ)

Table 1 is the central result at the anchor. At 0.41 effective bits per weight, all three post-hoc routes land above 6.3 bits per byte, most of the way to the random-byte floor of 8.0, while native QAT holds at 3.07. They differ from each other by less than a quarter of a bit per byte. That the published solver lands with our own implementation, rather than between it and native QAT, is the point: the failure below one bit is a property of the post-hoc routes we test at this scale, not of any one solver's tuning. The calibration-PTQ gap runs from 2.1 to 3.4 bits per byte, and the separation is complete:

in every one of the six seeds, native QAT scores below post-hoc quantization of that seed's own model, at all three bit-widths and for all three post-hoc classes (Figure 1). Even strong uniform PTQ at 2 bits (3.45 bpb, Appendix G) is beaten by native QAT at barely over one effective bit (1.09 bpw, 2.88 bpb).

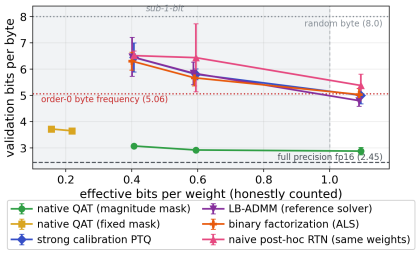

Figure 1: Quality versus counted effective bit-width at the 0.86M anchor (enwik8, 6 runs, error bars ± 1 std; series labeled in the legend). The three post-hoc classes of Table 1 (strong calibration PTQ, LB-ADMM of Chong et al. [2026], binary factorization) and naive RTN, the baseline they are meant to beat, all sit at or above the order-0 byte-frequency floor (5.06 bpb, red dotted) below one bit, worse than predicting byte frequencies, while native QAT stays near 3. The fixed-mask arm carries near-zero index cost (Appendix J). Shaded region is sub-1-bit.

3.2 Can a fine-tune close the gap? Only at training-scale cost

An immediate objection is that the post-hoc methods are denied any adaptation: they project a trained model into the sub-1-bit container and stop. We test a quantization-aware fine-tune of the strong calibration PTQ weights at two budgets, 5M tokens (20% of the original) and the full 25M. The fine-tune re-solves the surviving values and is free to update the survivor support (the $n{:}m$ mask is recomputed from the current weights at each step), so it is a warm-started QAT run rather than a constrained correction; Section 3.3 pins these ingredients separately. Each width and budget brackets its own rate, interior except the short-budget binary cell, a grid-edge bound (Appendix C).

The two budgets give different answers, and the difference is the finding (Table 5, in Appendix E). At 20% of the original tokens, binary recovers fully while both sub-1-bit widths retain a deficit of about 0.3 bits per byte. At the full budget the deficit closes and slightly reverses at every width, and at binary the fine-tuned model lands inside the fp16 anchor's own seed band. The approach is smooth rather than saturating (Appendix E), so the 0.3 bpb deficit a short-adaptation experiment measures is a budget effect, not a floor.

The compute accounting is two-sided. Two paths reach sub-1-bit quality at this scale: train natively once (25M tokens), or train dense and re-train quantized for as long again (50M total, 0.09 to 0.38 bpb better, worth it only if the dense model already exists). What we do not find is a cheap one. That is a statement about the routes on our bench, not a proof that none exists: a method that performs the optimization we isolate below at lower cost is exactly what our account predicts should work.

3.3 Why: the chasm is the missing re-solve, not the representation

The deployed sub-1-bit weight is a support and a set of values, $\mathrm{mask} \cdot \mathrm{sign}(W) \cdot s_{\mathrm{row}}$; the same container that is degenerate when projected into is usable when trained into. We decompose the gap at the 0.41-bpw anchor by returning one ingredient of training at a time to the post-hoc model, holding architecture, data, adaptation budget (25M tokens, equal to pretraining), and bit accounting fixed, each arm at a bracketed interior-optimum rate (Appendix C). The decomposition runs in the fine-tune harness of Section 3.2. Its project-and-stop cells average 6.41 ± 0.29 bits per byte (31 measurements over three pretrain seeds; the contributing arms and the case for pooling them are given in Appendix C), consistent with the anchor's 6.447 ± 0.551. The from-scratch reference (3.073 ± 0.060) is the anchor value of Table 1. We test four candidate explanations. Each is decided by a contrast we fixed before its deciding run, and the support contrast carried a numeric falsification threshold set in advance. The results falsify three of the four, the survivor support, the initialization and the calibration estimate, and leave the fourth standing, the missing re-solve against the deployed representation (Figure 3).

Not the survivor selection. The natural suspicion is that post-hoc methods keep the wrong weights. They do not; at this anchor it barely matters which weights the projection keeps. Pinning the survivor mask to exactly the positions strong calibration PTQ chose, freezing it, and re-solving only the values through the straight-through estimator recovers 3.115 ± 0.015 bits per byte, within 0.05 of from-scratch QAT (3.073 ± 0.060). Replacing that mask with a uniformly random one, matched in every other respect, lands at 3.150 ± 0.012 (three seeds each): once the values may adapt, the two supports are separated by less than our run-to-run floor (Appendix B), against more than 2.5 bits per byte when they may not (the random projection with frozen values scores 9.1, the calibration-chosen one 6.4). Survivor selection looks decisive only from inside the post-hoc regime, where nothing can move, a contingency the pruning literature predicts [Renda et al., 2020, Liu et al., 2022a]; a data-blind fixed mask still reaches 3.72 bits per byte at 0.156 bpw (Appendix J). Nor does the mask travel: transplanting a QAT model's survivor positions onto full-precision values scores worse than naive RTN, and a full-precision magnitude mask on QAT's values is indistinguishable from a random one (Appendix F). The advantage is a joint property of the trained matrix, not a transferable choice of which positions to keep.

Not the initialization, and not the calibration estimate. Starting the value re-solve from the post-hoc solution rather than the trained dense weights separates the arms by less than that same floor (3.148 ± 0.046 against 3.115 ± 0.015, three seeds each, every arm at its own bracketed rate): the initialization is not resolved either. Nor is this an estimation problem: growing the calibration set from 50k to 5M tokens collapses the run-to-run spread of strong PTQ at 0.41 bpw from ± 0.23 to ± 0.004 bits per byte, but buys no accuracy, the three-draw mean wandering without trend between 6.43 and 6.64 and ending at 6.46, past the order-0 floor throughout. The Hessian estimate converges; the converged answer remains degenerate.

The layer-local objective decouples from the loss below one bit. Calibration PTQ minimizes per-layer reconstruction error, a proxy faithful only while quantization is a small perturbation [Frantar et al., 2022]; zeroing fifteen of sixteen weights is not one. Tracing both quantities down the ladder on one trained model (Appendix G), the proxy and the loss move together through GPTQ 8/4/3/2-bit, then separate below one bit to the point of inversion at 1:16, where the strong method's uniformly lower per-layer reconstruction error lands *above* RTN in bits per byte in every draw. Optimizing the local proxy harder is not a path out of the collapse.

What is sufficient: re-solving against the deployed weights, and remarkably little of it. The counterpart to those negatives is how cheap the repair is in parameters, though not in tokens. Keeping every quantized weight bit-frozen, exactly as strong PTQ shipped it, and retraining only the full-precision interface, the token and position embeddings, layer norms, and biases, 8.4% of parameters, recovers $6.41 \pm 0.29 \rightarrow 3.17 \pm 0.04$: about 97% of the collapse, within 0.10 of from-scratch QAT. The two halves of the interface are complementary: norms and biases alone reach 3.85 ± 0.03 and embeddings alone 3.37 ± 0.01.

The arm replicates away from the anchor: repeated at the 4.87M point, at its own bracketed rate and three seeds, it recovers 89% of the collapse rather than 97% (Appendix K). Most of the collapse closes without re-solving a single quantized value at both configurations, though the recovered fraction is not constant across them.

A repair touching 8.4% of parameters would be a cheap post-hoc path if it also needed few tokens. It does not. Sweeping the adaptation budget from 0.5M to 25M tokens, each at its own bracketed rate (Appendix E), the repair improves monotonically with no knee to exploit, and the full budget does not overtake from-scratch QAT. The repair is cheap in parameters and expensive in tokens: what buys quality is training-scale optimization against the deployed weights, not the number of parameters allowed to move.

This scale of repair is qualitatively beyond the fractions of a bit that norm or auxiliary tuning buys at 2 to 4 bits [Li et al., 2023, Tseng et al., 2024, Egiazarian et al., 2024], and a representation-level probe (Appendix I) shows why: what the interface performs is a global re-coding into the frozen quantized stack, not a per-channel correction of it.

3.4 The gap persists under token-matched scaling

Everything so far is one model size, at about 29 tokens per parameter. A mechanism shown at a single point might belong to that point, so we ask whether the gap survives as the model grows. To separate scale from a particular training budget, we run a token-matched sweep: a uniform-depth (L=6) ladder from 1.26M to 19.18M parameters, each model trained at a fixed 18 tokens per parameter on byte-level enwik9 (a superset of enwik8), three seeds per point, each arm at its own bracketed best learning rate. Fixing tokens per parameter removes the confound of a fixed-token budget, where a larger model is more undertrained [Ouyang et al., 2025], so a trend in the gap reflects scale rather than training budget. The sweep's 18 tokens per parameter sits below the anchor's 29, and undertrained models are the easier case for post-hoc quantization, so the budget choice, if anything, favors PTQ.

Table 2: Gap versus scale under a token-matched budget (enwik9, fixed 18 tokens per parameter, uniform L=6, three seeds per arm, each arm at its own bracketed best learning rate). Gap is strong calibration PTQ minus native QAT in bits per byte at matched effective bit-width. In all fifteen cells the separation is complete: native QAT scores below post-hoc quantization in every seed, at every width, at every scale. Effective bit-width shrinks with scale as the per-row scale overhead amortizes over more input features: binary 1.09 to 1.02, 1:8 0.59 to 0.52, and 1:16 0.41 to 0.34 bits per weight.

scale	binary	1:8	1:16
1.26M	+2.25	+2.89	+3.37
2.77M	+2.82	+3.75	+4.32
4.87M	+2.84	+3.93	+4.29
10.84M	+2.73	+3.53	+4.24
19.18M	+2.83	+3.73	+4.36

The gap does not close. Across a 15× range of parameters, native QAT beats strong calibration PTQ by 2.2 to 4.4 bits per byte at every scale and bit-width (log-log slope +0.06 for binary, +0.07 for both sub-1-bit arms). The upward slope needs one caveat. Below one bit, strong calibration PTQ is already at or past the order-0 byte-frequency floor of the enwik9 validation split (5.15 bpb) at every scale, while native QAT stays well below it (1.4 to 3.0 bpb) and keeps improving as the model grows, so part of the slope reflects PTQ stuck at that degenerate floor rather than a growing training-path cost; binary, at about one bit, stays below the floor. The sub-1-bit budgets tighten with scale rather than loosen, so the held gap is conservative.

We read this as a property of the sub-1-bit regime, not of one token budget. The sweep does not vary parameter redundancy at fixed training, so we make no claim that redundancy is what post-hoc quantization exploits at large scale. What it settles is a question of baseline: the common report that post-hoc quantization is nearly free at scale is measured against the full-precision model; measured instead against a model trained natively at the same bit-width, the cost of rounding does not shrink across the scales we test. A single-seed PyTorch/CUDA rerun tracks the same persistence out to about 43M parameters, making an MLX-specific artifact unlikely. Native QAT does pay a real precision tax against its own fp16 anchor, 17 to 25% in bits per byte (Appendix P), but the chasm of Table 1 is several times larger than that.

4 Related Work, Scope, and Limitations

Related work. The sub-1-bit regime is reached today almost entirely from pretrained full-precision weights: structured binary PTQ at 0.55 bits per weight [Dong et al., 2024], latent binary factorization to binary and below [Chong et al., 2026] and as deep as 0.1 bits per weight [Lee et al., 2025], and codebook methods at 0.7 to 1.11 bits [Gu et al., 2025]. Native low-bit *training* exists but floors at or above one bit [Wang et al., 2023, Ma et al., 2024, Zhang et al., 2026], and the two scaling studies that map the low-bit regime stop at or above one bit and far above our parameter range [Liu et al., 2025, Kaushal et al., 2024]. The comparison this paper makes, post-hoc and native paths into the *same* sub-1-bit container at matched effective bit-width, is one we have not found reported: the two literatures are separated by the scale at which each is practical.

The deepest of those routes enter the container the way we find necessary, and one does not. LittleBit [Lee et al., 2025] reaches 0.1 bits per weight by training its factors rather than projecting them, and

LittleBit-2 [Lee and Kim, 2026] refines only the *initialization* of that pipeline, still learning the factors under quantization-aware training. STBLLM [Dong et al., 2024] is the exception, and it bounds our claim: it binarizes in one shot, with gradient-free importance metrics and explicitly no fine-tuning, and at 0.55 bits per weight it still holds a 7B model together (31.72 perplexity, against 688.73 for BiLLM [Huang et al., 2024] at the same width). A projection-only route can therefore reach sub-1-bit and remain a language model, so what we isolate is not a law about the container but a statement about what a projection has left to work with. STBLLM's own ladder reads that way: the identical 0.55-bit method scores 11.07 at 65B, improving as redundancy grows, and its method class, error compensation over an $n{:}m$ mask, is the class of our strong calibration baseline, which at 0.86M parameters lands past the byte-frequency floor (Appendix Q).

Dremov et al. [2026] split a fixed budget between a full-precision phase and a QAT phase at 1 to 6 bits and 86M to 2.2B parameters. We observe their ordering, a warm-started QAT phase beating QAT alone, below one bit as well (Section 3.2), though directionally only: they hold total compute fixed, while our warm-started arm spends twice the tokens of the from-scratch one. Their ladder bottoms out at one bit and compares QAT against QAT, so whether the post-hoc route reaches the container at all is adjacent to their setting rather than inside it (Appendix Q).

The quantization scaling laws [Kumar et al., 2025, Ouyang et al., 2025] predict a post-hoc penalty growing as models shrink and as they are more fully trained, and our token-matched sweep does not escape it: their law is fit in absolute size and tokens, not in the ratio we hold fixed. What it has no term for is a native arm, which sits in the same container at the same width and does not degrade at all (Appendix Q). We introduce no new encoding, sparsity scheme, or calibration routine: the contribution is the controlled comparison of training paths at matched effective bit-width, and the decomposition of the resulting gap, under a transparent bit accounting, on MLX [Hannun et al., 2023]. Appendix Q expands each of these lines.

Scope and limitations. Post-hoc quantization is represented here by three solvers, one of them Chong et al. [2026]'s own LB-ADMM, so the failure below one bit is not an artifact of a baseline we wrote ourselves. It remains a statement about solvers rather than pipelines: we do not reproduce the multi-stage billion-scale systems built around them, and a pipeline adding staged reconstruction or logit-level alignment is already doing some of the training-scale optimization our decomposition identifies as the active ingredient, so we would expect it to close part of this gap, at a cost that is no longer post-hoc. Our fine-tune and decomposition experiments (Sections 3.2 and 3.3) all begin from the strong calibration PTQ solution, so they establish that route's sub-1-bit deficit is repairable by training-scale optimization; we do not separately test whether the factorization or LB-ADMM solutions repair as readily. Experiments are byte-level and remain small-scale and quality-only, though a BPE stress test (Appendix L) checks the tokenization and two further corpora (Appendix M) check the domain. We leave the matched-memory question open: at this scale, a native sub-1-bit model against a smaller full-precision one at equal bytes is confounded by how well the small baseline is tuned. No fast sub-1-bit kernel exists on this backend, so the $n{:}m$ weights dequantize to a dense forward pass and the regime buys storage rather than speed at these sizes (Appendix O), and the storage win must be quoted end to end: block linears shrink $39\times$, but the full-precision interface still ships, so the deployed model shrinks $9.3\times$ and peak inference memory stays dense-equivalent (Appendix N).

5 Conclusion

Below one bit, one-shot post-hoc projection does not return a degraded language model but one that falls to or past the order-0 byte-frequency floor, for all three solvers we test, the reference solver included. The repair is the step the compression literature identified above one bit, training-scale optimization against the deployed representation, and it works by each route we test, including retraining the 8.4% full-precision interface around bit-frozen weights. We claim neither that condition nor its order-agnosticism as new, only that it is what still separates a working model from a byte-frequency table below one bit, a regime in which it had not been tested. A plausible ceiling is scale, which would cut against a reading of this as a law about the container: a one-shot projection keeps a 7B model together below one bit where redundancy is abundant (Section 4), and at 0.86M parameters there is none to spare. Where the boundary between those two regimes falls is the question our range poses and cannot close.

References

Afsara Benazir and Felix Xiaozhu Lin. Profiling large language model inference on Apple Silicon: A quantization perspective. *arXiv preprint arXiv:2508.08531*, 2025.

Yoshua Bengio, Nicholas Léonard, and Aaron Courville. Estimating or propagating gradients through stochastic neurons for conditional computation. *arXiv preprint arXiv:1308.3432*, 2013.

Hyochan Chong, Dongkyu Kim, Changdong Kim, and Minseop Choi. NanoQuant: Efficient sub-1-bit quantization of large language models. In *International Conference on Machine Learning (ICML)*, 2026. arXiv:2602.06694.

Peijie Dong, Lujun Li, Yuedong Zhong, Dayou Du, Ruibo Fan, Yuhan Chen, Zhenheng Tang, Qiang Wang, Wei Xue, Yike Guo, and Xiaowen Chu. STBLLM: Breaking the 1-bit barrier with structured binary LLMs. *arXiv preprint arXiv:2408.01803*, 2024.

Aleksandr Dremov, David Grangier, Angelos Katharopoulos, and Awni Hannun. Compute-optimal quantization-aware training. In *International Conference on Learning Representations (ICLR)*, 2026. arXiv:2509.22935.

Vage Egiazarian, Andrei Panferov, Denis Kuznedelev, Elias Frantar, Artem Babenko, and Dan Alistarh. Extreme compression of large language models via additive quantization. *arXiv preprint arXiv:2401.06118*, 2024.

Ronen Eldan and Yuanzhi Li. Tinystories: How small can language models be and still speak coherent english? *arXiv preprint arXiv:2305.07759*, 2023.

Steven K. Esser, Jeffrey L. McKinstry, Deepika Bablani, Rathinakumar Appuswamy, and Dharmendra S. Modha. Learned step size quantization. *International Conference on Learning Representations (ICLR)*, 2020.

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.

Hao Gu, Lujun Li, Hao Wang, Lei Wang, Zheyu Wang, Bei Liu, Jiacheng Liu, Qiyuan Zhu, Sirui Han, and Yike Guo. BTC-LLM: Efficient sub-1-bit LLM quantization via learnable transformation and binary codebook. *arXiv preprint arXiv:2506.12040*, 2025.

Awni Hannun, Jagrit Digani, Angelos Katharopoulos, and Ronan Collobert. MLX: Efficient and flexible machine learning on apple silicon. `https://github.com/ml-explore/mlx`, 2023.

Wei Huang, Yangdong Liu, Haotong Qin, Ying Li, Shiming Zhang, Xianglong Liu, Michele Magno, and Xiaojuan Qi. BiLLM: Pushing the limit of post-training quantization for LLMs. *arXiv preprint arXiv:2402.04291*, 2024.

Ayush Kaushal, Tejas Pandey, Tejas Vaidhya, Aaryan Bhagat, and Irina Rish. Spectra: A comprehensive study of ternary, quantized, and FP16 language models. *arXiv preprint arXiv:2407.12327*, 2024.

Tanishq Kumar, Zachary Ankner, Benjamin F. Spector, Blake Bordelon, Niklas Muennighoff, Mansheej Paul, Cengiz Pehlevan, Christopher Ré, and Aditi Raghunathan. Scaling laws for precision. *International Conference on Learning Representations (ICLR)*, 2025.

Banseok Lee and Youngmin Kim. LittleBit-2: Maximizing the spectral energy gain in sub-1-bit LLMs via latent geometry alignment. *arXiv preprint arXiv:2603.00042*, 2026.

Banseok Lee, Dongkyu Kim, Youngcheon You, and Youngmin Kim. LittleBit: Ultra low-bit quantization via latent factorization. *arXiv preprint arXiv:2506.13771*, 2025.

Hanyang Li, Jianhao Ma, and Ying Cui. Understanding quantization-aware training: Gradients at quantized weights bias to the low-loss basin. *arXiv preprint arXiv:2606.09012*, 2026.

Liang Li, Qingyuan Li, Bo Zhang, and Xiangxiang Chu. Norm tweaking: High-performance low-bit quantization of large language models. *arXiv preprint arXiv:2309.02784*, 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 LLM compression and acceleration. In *Proceedings of Machine Learning and Systems (MLSys)*, 2024.

Shiwei Liu, Tianlong Chen, Xiaohan Chen, Li Shen, Decebal Constantin Mocanu, Zhangyang Wang, and Mykola Pechenizkiy. The unreasonable effectiveness of random pruning: Return of the most naive baseline for sparse training. In *International Conference on Learning Representations*, 2022a.

Zechun Liu, Zhiqiang Shen, Marios Savvides, and Kwang-Ting Cheng. ReActNet: Towards precise binary neural network with generalized activation functions. *European Conference on Computer Vision (ECCV)*, 2020.

Zechun Liu, Barlas Oguz, Aasish Pappu, Lin Xiao, Scott Yih, Meng Li, Raghuraman Krishnamoorthi, and Yashar Mehdad. BiT: Robustly binarized multi-distilled transformer. *Advances in Neural Information Processing Systems (NeurIPS)*, 2022b.

Zechun Liu, Changsheng Zhao, Hanxian Huang, Sijia Chen, Jing Zhang, Jiawei Zhao, Scott Roy, Lisa Jin, Yunyang Xiong, Yangyang Shi, Lin Xiao, Yuandong Tian, Bilge Soran, Raghuraman Krishnamoorthi, Tijmen Blankevoort, and Vikas Chandra. ParetoQ: Improving scaling laws in extremely low-bit LLM quantization. *arXiv preprint arXiv:2502.02631*, 2025.

Shuming Ma, Hongyu Wang, Lingxiao Ma, Lei Wang, Wenhui Wang, Shaohan Huang, Li Dong, Ruiping Wang, Jilong Xue, and Furu Wei. The era of 1-bit LLMs: All large language models are in 1.58 bits. *arXiv preprint arXiv:2402.17764*, 2024.

Matt Mahoney. Large text compression benchmark. `http://mattmahoney.net/dc/text.html`, 2011.

Jinendra Malekar and Ramtin Zand. Amdahl's law for LLMs: A throughput-centric analysis of extreme LLM quantization. *Transactions on Machine Learning Research*, 2025. URL `https://openreview.net/forum?id=JtrQJJQYpP`.

Xu Ouyang, Tao Ge, Thomas Hartvigsen, Zhisong Zhang, Haitao Mi, and Dong Yu. Low-bit quantization favors undertrained LLMs: Scaling laws for quantized LLMs with 100t training tokens. *Annual Meeting of the Association for Computational Linguistics (ACL)*, 2025.

Alex Renda, Jonathan Frankle, and Michael Carbin. Comparing rewinding and fine-tuning in neural network pruning. *arXiv preprint arXiv:2003.02389*, 2020.

Albert Tseng, Jerry Chee, Qingyao Sun, Volodymyr Kuleshov, and Christopher De Sa. Quip#: Even better LLM quantization with hadamard incoherence and lattice codebooks. *arXiv preprint arXiv:2402.04396*, 2024.

Moritz Wagner, Christophe Roux, Max Zimmer, and Sebastian Pokutta. A free lunch in LLM compression: Revisiting retraining after pruning. *arXiv preprint arXiv:2510.14444*, 2025.

Hongyu Wang, Shuming Ma, Li Dong, Shaohan Huang, Huaijie Wang, Lingxiao Ma, Fan Yang, Ruiping Wang, Yi Wu, and Furu Wei. BitNet: Scaling 1-bit transformers for large language models. *arXiv preprint arXiv:2310.11453*, 2023.

Di Zhang, Xun Wu, Shaohan Huang, Yudong Wang, Hanyong Shao, Yingbo Hao, Zewen Chi, Li Dong, Ting Song, Yan Xia, Zhifang Sui, and Furu Wei. Sparse-BitNet: 1.58-bit LLMs are naturally friendly to semi-structured sparsity. *arXiv preprint arXiv:2603.05168*, 2026.

A Experimental design and bit-accounting

Naive RTN at the anchor. Table 1 reports the three real post-hoc solvers; round-to-nearest is the baseline they are meant to beat, and at the anchor it scores 5.379 ± 0.433 (binary), 6.438 ± 1.292 (1:8) and 6.515 ± 0.184 (1:16) bits per byte over the same six seeds. RTN is therefore the one route already past the order-0 floor at 1.09 bits, where the three real solvers still clear it: rounding alone gives up the language model a full bit-width before the sub-1-bit boundary. Against those, strong calibration PTQ closes 15%, 18% and 2% of the distance from RTN to native QAT, which is the 2 to 18% quoted in Appendix G. The upper end of that range is not robust: the 1:8 RTN mean is pulled up by a single seed scoring 8.99, and on the median (6.00) the closure at 1:8 is 6%, not 18%. Strong calibration still beats RTN at 1:8 on either statistic. The honest reading of the range is the one the body gives it, that the strong method closes a small part of the RTN gap and leaves the collapse intact, and not a precise closure figure.

Rank accounting for the factorization arm. The per-layer rank r is the largest whose transparent cost, $r(\text{out}+\text{in})$ sign bits plus $16r$ fp16 scale bits over out $\times$ in, fits the target bit-width, so rank is set by the budget and not chosen to flatter the baseline. Each accepted coordinate-descent flip strictly lowers reconstruction error, and we run four sweeps, fixed before any result was seen. That constant is not what puts the arm where it is: extending the schedule to 32 sweeps drives the solver's own reconstruction objective monotonically down while leaving the loss no better, and on one seed 0.7 bpb worse (Appendix H). Allowing full rank $r = \min(\text{out}, \text{in})$ removes the budget entirely and asks what the binary-factor representation can do at all: it lands $+2.17$ bpb above fp16, so the failure below one bit is not a rank shortage.

The Hessian-optimal per-row scale. The strong calibration arm of Section 3.1 scales each surviving row by the scalar that minimizes its calibration reconstruction error. Writing q_r for the signed survivor pattern of output row r (entries ± 1 on its surviving $n{:}m$ positions and 0 elsewhere), w_r for that row's full-precision weights, and H for the layer's calibration Hessian, the least-squares-optimal scalar is $s_r^* = (q_r^\top H w_r)/(q_r^\top H q_r)$. This is the best a per-row scale can do on the calibration objective for a given mask, which is what makes the arm a strong baseline on that objective and what makes its failure on the loss below one bit informative rather than a tuning artifact.

The per-row fp16 scale is the floor that stops the counted width from reaching 0.1 bpw at the anchor width: one 16-bit scale per row of 128 contributes $16/128 \approx 0.125$ bits per weight before a single sign bit is spent, which is why the 1:16 arm counts 0.41 bpw rather than 0.06. Amortizing one scale across G rows would push the floor down by the same factor and is future work; we keep the per-row scale here because it is the setting the post-hoc solvers we compare against also use, so the accounting stays symmetric across arms.

Table 3: Bit-accounting breakdown for the two $n{:}m$ patterns at the common embedding width ($d_{\text{in}} = 128$, representative of qkv/proj/fc linears). The feed-forward projection operates at $d_{\text{in}} = 512$, reducing its per-row scale cost to $16/512 = 0.03$ bits. Averaging each linear's bit-width by its parameter share yields the model-wide averages 0.59 (1:8) and 0.41 (1:16) reported in the text.

Component	1:8	1:16
Survivor sign bits (n/m)	0.125	0.0625
Mask index ($\log_2 \binom{m}{n}/m$)	0.375	0.250
Per-row scale ($16/d_{\text{in}}$)	0.125	0.125
Total at $d_{\text{in}} = 128$	0.625	0.4375
Param-weighted model-wide average	0.59	0.41

B Reproducibility: the full configuration

The run-to-run floor, and why the smallest deltas are reported as unresolved. Two of the contrasts in Section 3.3 are hundredths of a bit, so we measured this harness's own reproducibility rather than assume it. Re-running nine cells of that section at their shipped rates and seeds, changing nothing, reproduces them to within 0.050 bits per byte (median 0.022, n=9 paired replicates). MLX

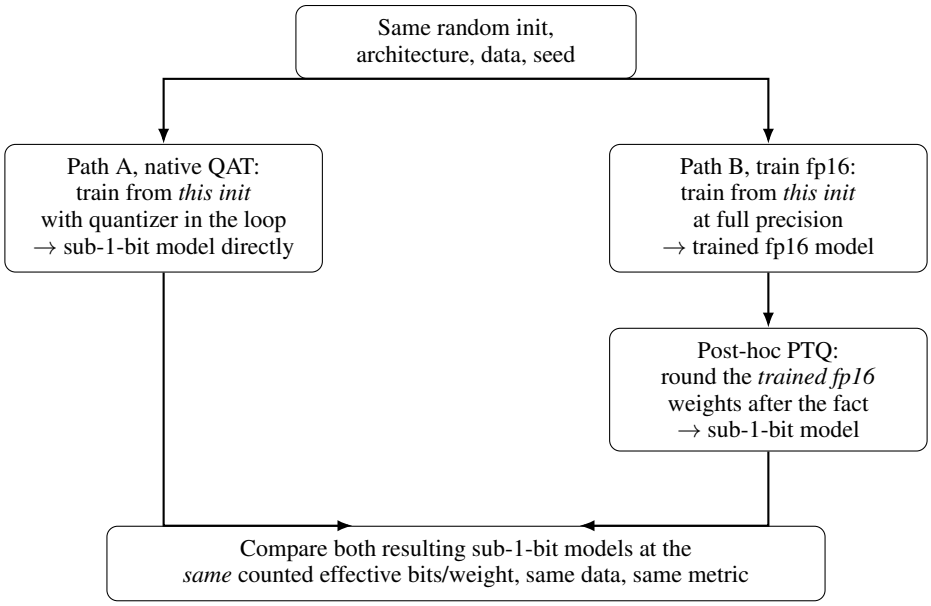

Figure 2: Experimental design. Native QAT and the post-hoc baselines start from the *same* random initialization, architecture, data, and seed as full precision; they do not share a common trained model. Path A trains directly with the quantizer in the loop and never exists in full precision. Path B first trains a separate full-precision model to convergence, which the post-hoc methods then round down after the fact. Only the resulting sub-1-bit models are compared, at matched effective bit-width.

kernels are not bit-deterministic across runs, and at this model size the resulting spread is larger than the support-content and initialization deltas themselves. We therefore report those two as unresolved from zero rather than as measured quantities. The contrast the section rests on is unaffected: support content is worth more than 2.5 bits per byte when the values cannot adapt, some fifty times this floor.

Why the fp16 reference differs by 0.003 bpb between tables. Every comparison in this paper is paired within a single run: the script that produces an arm also trains and evaluates the full-precision model it is compared against, on the same seed, data order, and initialization. The post-hoc pipeline and the QAT pipeline are separate scripts, so each carries its own fp16 anchor at the same bracketed rate. Table 1 and Table 13 quote the anchor paired with the post-hoc arms (2.449 ± 0.235, six seeds); Table 15 quotes the one paired with the QAT arms (2.452 ± 0.238, six seeds). The two agree to 0.003 bpb, two orders of magnitude below the effects we report. We keep them separate rather than pooling them because pairing, not the third decimal, is what the comparisons rest on.

Optimization. All arms, from-scratch and adapted, train with AdamW ($\beta_1 = 0.9$, $\beta_2 = 0.999$, $\epsilon = 10^{-8}$) at weight decay 0.1, batch 64, context 256, and no gradient clipping. From-scratch runs warm up linearly over $\max(10, \text{steps}/50)$ steps and then follow a cosine decay to 10% of the peak rate; the anchor's 25M-token budget is 1525 steps. Fine-tune and repair runs warm up over $\max(3, \text{steps}/10)$ steps and then hold the rate constant, which is why their selected rates are not comparable to from-scratch rates and are bracketed separately (Appendix C). Weights and biases initialize under MLX defaults with the run seed set explicitly, so a run reproduces exactly from its seed. Arithmetic is float32 throughout; `fp16` names a 16-bit storage budget, not a compute format, and the quantized arms differ from it only in the counted bit-width of the block linears.

Data. enwik8 is the first 10^8 bytes of the Wikipedia dump at its canonical split, the first 90M bytes for training and bytes 90M to 95M held out for validation. Batches are drawn uniformly at random from the training portion, so a 25M-token budget means 25M tokens seen ($1525 \times 64 \times 256$), sampled from the 90M available, not a contiguous prefix. GPTQ calibration Hessians are accumulated from the training portion only, under a fixed calibration seed. Validation bits per byte is the cross-entropy over 60 batches drawn from the held-out portion under a fixed evaluation seed shared by every arm,

so all arms are scored on identical validation batches; converting to bits per byte divides by $\log 2$ and by the corpus bytes per token (1.0 for byte-level corpora). The token-matched sweep uses enwik9, the first 10^9 bytes of the same dump, so that models up to 19.18M parameters can each take 18 tokens per parameter without repeating data. The second-corpus check (Appendix M) uses byte-level Python source and TinyStories under the identical protocol. All corpora are byte-level with a 256-symbol vocabulary, except the BPE stress test (Appendix L), whose tokenizer is trained on the enwik8 training portion alone at vocabulary 1000 and applied to both splits.

Corpora, splits, and licensing. Every corpus is public and is rebuilt from its source by the released scripts; we redistribute no corpus. enwik8 and enwik9 are the first 10^8 and 10^9 bytes of the English Wikipedia XML dump of 3 March 2006, obtained through the Large Text Compression Benchmark; the article text carries the source project's own terms and the benchmark distributes no separate license for the files. TinyStories is used at its published `train`/`validation` splits under CDLA-Sharing-1.0 [Eldan and Li, 2023]. The Python corpus is the standard library of the local CPython 3.11.7 (PSF License Agreement), `.py` files only, with `site-packages`, `lib-dynload`, and `__pycache__` excluded, files ordered by path and cut at a *file* boundary at 88% of bytes, so no file appears in both splits. The artifact ships a manifest of the Python version, the file list, and a per-file SHA-256, which reconstructs that corpus exactly on a matching interpreter.

Hardware and software. Every number in this paper comes from a single Apple M5 Max MacBook Pro (40-core GPU, 128 GB unified memory, macOS $26.4.1$) under MLX $0.31.2$ and Python 3.11, with the single exception of the PyTorch/CUDA cross-backend replication noted in Section 3.4. That replication is single-seed, and under its compute cap its full-precision anchor selects a learning rate at the edge of the swept grid, so it is a bound rather than a confirmed optimum and we read it only as a directional cross-backend check; an error-barred replication is future work. Its binary gap, strong calibration PTQ minus native QAT at matched tokens per parameter, reads $+2.33$, $+2.54$, $+2.71$, $+2.83$, $+3.53$, and $+3.64$ bits per byte at 1.26M, 2.77M, 4.87M, 10.8M, 29.9M, and 42.9M parameters. An anchor-scale cell trains in one to three minutes, which is what makes a per-arm bracketed learning-rate protocol affordable across every cell in the paper.

C Learning-rate brackets for the adaptation arms

Optima do not transfer across protocols: the best from-scratch rate, the best fine-tune rate, and the best repair rate at the same width differ by up to $16\times$. Every adapted cell in the paper is therefore bracketed within its own protocol, and a cell is only reported at a rate whose grid neighbors are both worse (a confirmed interior minimum), with the exceptions marked in place. Table 4 lists the grids. Extending the random-support grid moved its optimum from $8\mathrm{e}{-3}$ to $1.6\mathrm{e}{-2}$, so its seeds were re-run at the new rate; every reported mean uses the post-extension optimum. From-scratch rates are in Section 2 (anchor) and are re-bracketed per scale in Section 3.4. The constrained arms prefer markedly higher rates: the straight-through fine-tune selects $4\mathrm{e}{-3}$ at every width, the interface-only and split repairs $8\mathrm{e}{-3}$, and the pinned-support re-solves up to $1.6\mathrm{e}{-2}$.

Because the two arms of a contrast can select different rates, every delta in Section 3.3 compares each arm at its own selected rate, never both at a shared one. The initialization delta is the case where this matters: the post-hoc-initialized re-solve selects $1.6\mathrm{e}{-2}$ and scores 3.148 ± 0.046, the dense-initialized re-solve selects $8\mathrm{e}{-3}$ and scores 3.115 ± 0.015, a separation smaller than the run-to-run floor below. Reading both arms off the same rate instead would compare a tuned cell against an untuned one and inflate the delta.

Does the selected rate survive a disjoint-seed re-selection? The anchor rates were chosen on the same seeds the anchor means are reported over, so we re-ran the whole anchor grid as an independent replication and re-selected on seeds $\{0, 1, 2\}$ with $\{3, 4, 5\}$ held out. The fp16 baseline's rate comes back confirmed and unchanged: $2\mathrm{e}{-3}$ beats both $1\mathrm{e}{-3}$ and $3\mathrm{e}{-3}$ on the selection half (2.410 against 2.486 and 2.560) and again on all six seeds (2.454 against 2.491 and 2.711). The precision tax is therefore measured against a baseline sitting at its own bracketed interior optimum, and every shipped cell reproduced to within the run-to-run floor of Appendix B.

The from-scratch anchor arms are the one family we do not annotate interior or edge in Table 4, and the reason is worth stating rather than leaving as an omission. Their grid differences are smaller than

their own run-to-run spread. Re-running the binary cell at a fixed seed and rate, changing nothing, has produced 2.76, 2.95 and 3.14 bits per byte across three executions, a range of 0.38: the sign and magnitude-ranking operations amplify the nondeterminism of GPU reductions far beyond the 0.05 floor the repair cells show (Appendix B). An interior-minimum mark on a grid whose steps are a tenth of that would assert a resolution the harness does not have. This is why those arms ship as six-run sets rather than single cells, and why we tune them by lowest mean over three seeds rather than by a seed-0 bracket. It also bounds what the choice can cost: the arms sit 2 to 3 bits per byte from the post-hoc routes they are compared against, so a rate ambiguity of a few tenths cannot reach the comparison.

What the replication also shows is that the argmin itself is not identifiable at this scale. The re-selected rate disagrees with the shipped rate for the three quantized arms while fp16 keeps its rate, and its two halves disagree for fp16, binary and $1{:}8$ while $1{:}16$'s halves agree: the two triples are different arms, so no single rate is preferred under every split. The $1{:}16$ case is representative: the replication prefers $4\mathrm{e}{-3}$ over the shipped $5\mathrm{e}{-3}$ by 0.039 bits per byte, while the original bracket prefers $5\mathrm{e}{-3}$ by 0.049, both margins inside the run-to-run floor. Because our kernels are not bit-deterministic, fixing a seed does not fix a run, so we keep each arm at the rate its own bracket selected rather than mix two run families in one table. The residual ambiguity is bounded: within every quantized arm the shipped and re-selected rates differ by no more than 0.05 bits per byte, against post-hoc gaps of more than 2.

The project-and-stop reference, run by run. Section 3.3 quotes a pooled project-and-stop value of 6.41 ± 0.29 bits per byte. This pools every measurement in the mechanism harness of the quantized model evaluated on the calibration-chosen support before any re-solve: 31 evaluations across three pretrain seeds, contributed by the pinned-support, post-hoc-initialized, co-adapted, norms-only, embeddings-only, and forensics arms, each of which measures the same quantity because each begins from the same projection. The arms differ only in what happens afterwards, so their pre-repair evaluations are repeated measurements of one number, and the spread is seed variance plus the nondeterminism of GPU reductions. We pool all of them rather than a chosen subset. The random-support arms are excluded because they project onto a different support and measure a different quantity (9.10 ± 0.99, $n{=}10$). `v4_stats.py` recomputes both pools from the run files.

Table 4: Fine-tune and repair learning-rate grids (0.86M anchor; grids log-spaced; LR* = selected rate; "interior" means both tested neighbors are worse at seed 0).

arm	tokens	tested grid	LR*
PTQ+FT binary	5M	$3\mathrm{e}{-5}$ to $5\mathrm{e}{-4}$	$5\mathrm{e}{-4}$ (edge: a bound)
PTQ+FT $1{:}8$	5M	$3\mathrm{e}{-4}$ to $4\mathrm{e}{-3}$	$1\mathrm{e}{-3}$ (interior)
PTQ+FT $1{:}16$	5M	$3\mathrm{e}{-4}$ to $8\mathrm{e}{-3}$	$4\mathrm{e}{-3}$ (interior)
PTQ+FT binary	25M	$2.5\mathrm{e}{-4}$ to $8\mathrm{e}{-3}$	$4\mathrm{e}{-3}$ (interior)
PTQ+FT $1{:}8$	25M	$5\mathrm{e}{-4}$ to $8\mathrm{e}{-3}$	$4\mathrm{e}{-3}$ (interior)
PTQ+FT $1{:}16$	25M	$1\mathrm{e}{-4}$ to $1.6\mathrm{e}{-2}$	$4\mathrm{e}{-3}$ (interior)
interface-only repair	25M	$2\mathrm{e}{-3}$, $8\mathrm{e}{-3}$, $3.2\mathrm{e}{-2}$	$8\mathrm{e}{-3}$ (interior, $4\times$ steps)
norms+biases only	25M	$2\mathrm{e}{-3}$, $8\mathrm{e}{-3}$, $3.2\mathrm{e}{-2}$	$8\mathrm{e}{-3}$ (interior, $4\times$ steps)
embeddings only	25M	$2\mathrm{e}{-3}$, $8\mathrm{e}{-3}$, $3.2\mathrm{e}{-2}$	$8\mathrm{e}{-3}$ (interior, $4\times$ steps)
re-solve, pinned PTQ support	25M	$2\mathrm{e}{-3}$ to $3.2\mathrm{e}{-2}$	$8\mathrm{e}{-3}$ (interior)
re-solve, random support	25M	$2\mathrm{e}{-3}$ to $3.2\mathrm{e}{-2}$	$1.6\mathrm{e}{-2}$ (interior)
re-solve, PTQ init	25M	$3\mathrm{e}{-4}$ to $3.2\mathrm{e}{-2}$	$1.6\mathrm{e}{-2}$ (interior)

D The decomposition in one figure

Figure 3 collects the arms of Section 3.3 on one axis: the project-and-stop baseline, the two adaptations that re-solve against the deployed representation, and the controls that vary support content, initialization, and calibration quality while holding the re-solve fixed. The sections that follow give each arm its own protocol and raw numbers.

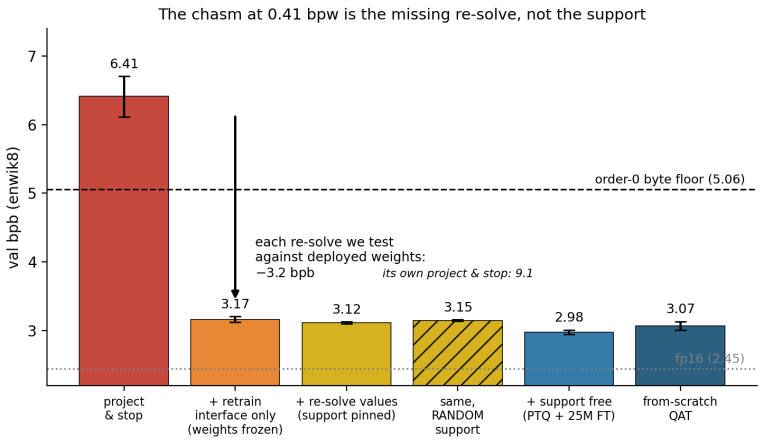

Figure 3: Decomposing the chasm at 0.41 bpw (enwik8, 0.86M anchor; every adapted arm gets 25M adaptation tokens at its own bracketed learning rate; mean $\pm$ std, $n{=}3$ per adapted arm, $n{=}31$ pooled project-and-stop measurements, $n{=}6$ from-scratch). The random-support arm (hatched) starts from its own project-and-stop at 9.10 ± 0.99 bpb ($n{=}10$) and lands beside the calibrated support. The fp16 line is the anchor's six-seed baseline (Table 1). Every re-solve we test against the deployed representation closes the gap; support content, initialization, and calibration quality do not move it beyond the run-to-run floor (Appendix B).

E Adaptation budgets: fine-tuning the projection, and repairing the interface

Two arms in the main text adapt a projected model rather than train one from scratch, and both are budget questions. Table 5 gives the quantization-aware fine-tune of the strong-PTQ weights at two budgets, discussed in Section 3.2. The rest of this appendix measures the cheaper of the two repairs, the interface-only one, across the full budget range.

Table 5: Strong PTQ + QAT fine-tune at two token budgets, each cell at its own bracketed best fine-tune learning rate (enwik8, 0.86M anchor; fine-tune cells $n{=}3$, native QAT $n{=}6$; mean $\pm$ std bits per byte).

eff. bpw	native QAT	strong calib. PTQ	+FT 5M (20%)	+FT 25M (100%)
0.41 (1:16)	3.073 ± 0.060	6.447 ± 0.551	3.348 ± 0.05	2.979 ± 0.033
0.59 (1:8)	2.921 ± 0.046	5.806 ± 0.204	3.200 ± 0.07	2.821 ± 0.042
1.09 (binary)	2.879 ± 0.141	5.004 ± 0.316	2.877 ± 0.10	2.503 ± 0.031

Section 3.3 shows that retraining only the full-precision interface (8.4% of parameters) around bit-frozen quantized weights recovers most of the collapse. If that repair were also cheap in tokens it would amount to a cheap post-hoc path, and the paper's central claim would be a statement about inconvenience rather than about training. This appendix measures it.

We sweep the adaptation budget from 0.5M to 25M tokens with the arm otherwise held fixed: support and initialization from the strong-PTQ solution, quantized weights bit-frozen, only the interface adapting. Optima do not transfer across budgets, so each budget brackets its own learning rate at seed 0 over $\{5\text{e}{-4}, 1\text{e}{-3}, 2\text{e}{-3}, 8\text{e}{-3}, 3.2\text{e}{-2}, 6.4\text{e}{-2}\}$ and is then run at three seeds at its winner. Every budget in Table 6 selects an interior minimum, with both neighbors worse; the short budgets prefer rates about four times lower than the full budget, which is why the grid extends below the rates used elsewhere in the paper.

The curve falls monotonically and shows no knee. The first halving of the gap to from-scratch QAT is the expensive one, costing about sevenfold in tokens; each later halving arrives faster, at closer to twofold. Even the full budget leaves the repaired model 0.09 bpb above native QAT, a margin of about 1.7 pooled standard deviations, so the repair approaches native quality without overtaking it.

Table 6: Interface-only repair versus adaptation budget (enwik8, 0.86M anchor, 0.41 bpw, n=3, mean $\pm$ std bits per byte). Every cell sits at its own bracketed interior optimum. References: from-scratch native QAT 3.073 ± 0.060 (n=6); the unrepaired strong-PTQ projection 6.41 ± 0.29 (n=31 measurements over three independent pretrain seeds).

adaptation budget	% of the native-QAT budget	bracketed LR	bits per byte
0.5M	2%	2e−3	4.153 ± 0.066
1M	4%	1e−3	3.933 ± 0.053
2.5M	10%	2e−3	3.740 ± 0.036
5M	20%	2e−3	3.506 ± 0.054
10M	40%	8e−3	3.300 ± 0.044
25M	100%	8e−3	3.165 ± 0.041

The practical reading is that the interface repair is cheap in parameters and expensive in tokens: it does not supply a shortcut that avoids training-scale optimization, it relocates that optimization into a smaller parameter set.

F Where the sub-1-bit advantage lives: support and values are co-adapted

A natural worry is that native QAT's edge is trivial: perhaps it lands on the same survivor positions a post-hoc method would, and the only difference is that QAT gets to retrain the surviving values. A transplant probe rules this out. At the d=128, L=4 anchor we train a full-precision model and a native-QAT model from the same seed, then build hybrid weights that take the survivor mask from one model and the surviving values (signs and per-row scale) from the other, holding the backbone fixed. Two diagonal cells reproduce the true baselines and gate the harness: mask and values both from QAT recover native QAT to 0.00 bits per byte, and both from full precision recover naive round-to-nearest (RTN). We report the 1:16 arm; 1:8 agrees (values in parentheses below). Transplanting QAT's mask onto full-precision values scores 9.07 bits per byte, worse than naive RTN at 6.31: the learned support is meaningless away from the values it co-adapted with. Importing a full-precision magnitude mask onto QAT's values scores 6.25, statistically indistinguishable from a random n:m mask on the same values at 6.36: a foreign model's magnitude ranking cannot locate QAT's survivors. The support is not irrelevant to magnitude in general, since QAT's own magnitude mask beats the random control by more than three bits per byte, but it is specific to the matrix that produced it. The sub-1-bit advantage is a joint property of the trained weight matrix, not a transferable choice of which positions to keep. Numbers are means over three seeds; the 1:8 arm gives QAT-mask on fp16 values 8.37, fp16-mask on QAT values 6.74, random-mask on QAT values 6.70, native 2.95, and RTN 5.85.

G The layer-local objective decouples from the loss below one bit

How much the strong baseline buys over naive RTN. At binary the strong n:m variant beats RTN by 0.38 bpb in all six seeds. Below one bit the across-seed margin is no longer a reliable win: +0.63 at 1:8 and +0.07 at 1:16, with the sign flipping seed to seed at both widths. The strong method closes only 2 to 18% of the RTN gap, so for this class of layer-local objectives the remainder is the cost of the projection rather than a weak baseline. The decoupling below is why: optimizing the proxy harder stops tracking the loss.

Figure 4(a) traces the quantity calibration PTQ actually optimizes against the quantity that matters. Through the uniform GPTQ ladder the two move together: better per-layer reconstruction, better model. On the seed-0 model drawn here the uniform ladder reads 2.40, 2.43, 2.54, and 3.40 bits per byte at 8, 4, 3, and 2 bits; across all six pretraining seeds the 2-bit cell averages 3.45 at 2.09 effective bits per weight, which is the figure the body quotes. Below one bit they separate, and the direction of the separation depends on the width. At 1:8, strong calibration keeps both a lower reconstruction error and a lower bits per byte than naive RTN in all six calibration draws (5.593 ± 0.034 against 5.754), so the proxy still points the right way. At 1:16 it inverts: the strong method achieves about 22% lower per-layer error in every draw and lands above RTN in bits per byte in every draw (6.524 ± 0.094 against 6.365). The six draws split three at 200k and three at 1M calibration tokens, so the inversion is not a calibration-size artifact. This experiment uses the same quantizer as Table 1 but

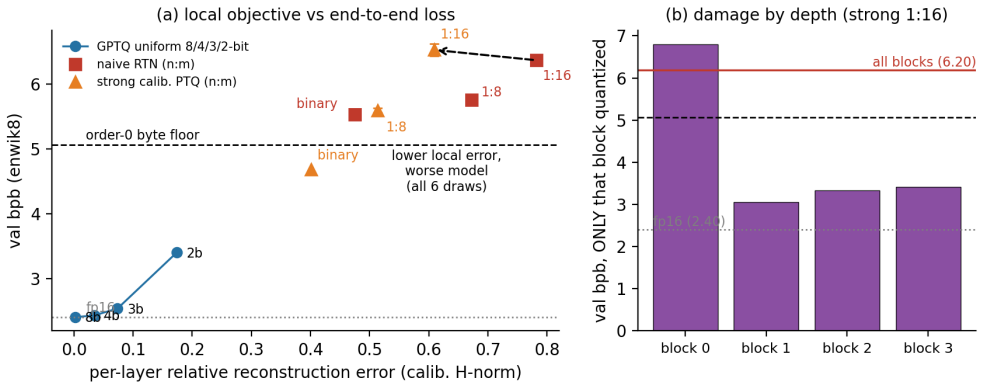

Figure 4: (a) Per-layer relative reconstruction error in the calibration H-norm (mean over the 16 block linears) against validation bits per byte, on one trained anchor model (seed 0). GPTQ uniform 8/4/3/2-bit lie on one smooth curve; the sub-1-bit $n{:}m$ points leave it, and at 1:16 the ordering inverts across six independent calibration draws (error bars ± 1 std over draws). (b) Quantizing one transformer block at a time (strong 1:16, everything else fp16, single eval): the damage concentrates in block 0.

isolates a different variance source: the table averages over six pretraining seeds (strong PTQ at 1:16 spans 5.86 to 7.30, standard deviation 0.55), while here one pretrained model is held fixed and only the calibration draw moves, which is why the two report 6.447 and 6.524 for the same arm. The across-seed number is the honest headline; the fixed-model number is what licenses a within-model comparison against RTN. An earlier single-draw 1:8 inversion did not survive this replication and is not claimed; the inversion we report is specific to 0.41 bpw.

Panel (b) locates the damage. With only block 0 quantized at strong 1:16 the model scores 6.80 bits per byte, worse than quantizing all four blocks at once (6.20), while blocks 1, 2, or 3 alone cost far less (3.06, 3.33, 3.42; fp16 is 2.40; single evaluation). The first block's projection error is amplified by the entire downstream network, which is consistent with the global, rather than layer-local, character of the failure: a per-layer objective is least faithful exactly where the network compounds its mistakes the most.

H Is the factorization baseline simply under-refined?

The binary-factorization arm runs its coordinate-descent refinement for four sweeps, a constant fixed before any result was seen. A fair objection is that the arm might then be reporting an under-refined solver rather than a property of the factorization class. The objection is answerable with data: hold the anchor, the target bit-width, and every other choice fixed, and vary only the sweep count over $\{1, 2, 4, 8, 16, 32\}$, an eightfold extension beyond what we ship. We also record the solver's *own* objective, the relative Frobenius reconstruction error of the factorized weights against the full-precision ones, so that the objective and the loss can be read side by side.

Table 7: Refinement sweeps against the factorization's own objective and against the loss (0.86M anchor, enwik8, 1:16 at 0.41 bpw; the shipped schedule is four sweeps). The objective is the relative Frobenius error over the sixteen block linears, measured at seed 0. Refining harder improves the objective monotonically and does not improve the loss.

refinement sweeps	1	2	4	8	16	32
reconstruction error (seed 0)	0.800	0.780	0.766	0.759	0.756	0.754
val bpb, seed 0	5.88	5.93	5.91	5.87	5.68	5.94
val bpb, seed 1	5.18	5.32	5.26	5.55	5.63	5.88

The two rows move differently, and that is the result. The solver's objective falls monotonically with every extra sweep, from 0.800 to 0.754: the refinement is working, and it is converging, with the

last doubling buying 0.002. The loss does not follow it. Validation bits per byte is non-monotone in the sweep count on both seeds, wandering within 0.26 bpb at seed 0; and at seed 1 every longer schedule is *worse* than a single sweep, ending 0.70 bpb higher at thirty-two than at one, while the objective improves throughout. Nowhere on either curve does the factorization come near native QAT's 3.07, and every point on both curves sits at or above the order-0 byte-frequency floor of 5.06.

So the four-sweep schedule is not what puts this arm where it is. Refining the factorization's own objective eight times harder than we ship buys nothing in loss and on one seed costs 0.7 bits per byte. This is the same decoupling that Appendix G measures on the GPTQ ladder, arriving here through a different solver family: below one bit, a better fit to the layer-local surrogate is not a better model. It also scopes the arm honestly. The number we report bounds this implementation of binary factorization at this budget, and the bound does not tighten when the implementation is refined harder, though we cannot rule out that a different factorization scheme reaches further.

I Representation-level forensics of the interface-only repair

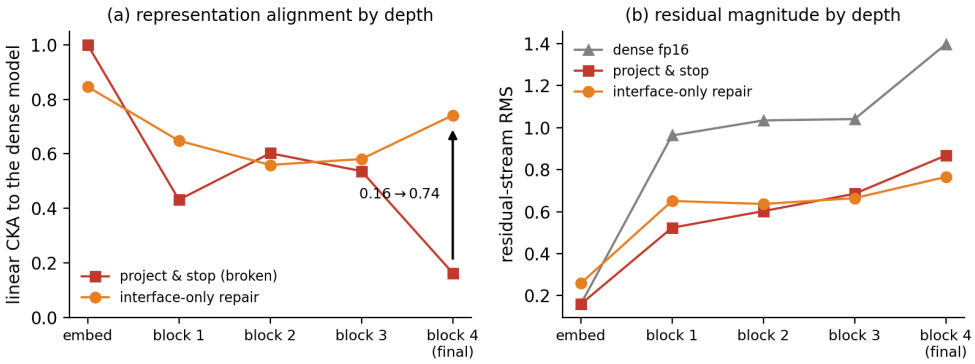

Figure 5: Representation forensics of the interface-only repair at strong $1{:}16$ (single seed; this seed scores fp16 2.40, project-and-stop 6.37, repaired 3.14). (a) Linear CKA between each model's residual stream and the dense model's, by depth. (b) Residual-stream RMS by depth.

Two facts about what the repair does. First, it realigns the output representation without retracing the dense computation: the project-and-stop model's final-layer CKA to the dense model is 0.16, the repaired model's is 0.74, yet middle-depth alignment stays middling in both models, 0.43 to 0.60 broken against 0.56 to 0.65 repaired (Figure 5a). Second, it works at the deflated scale the projection leaves behind: projection cuts every block's residual contribution roughly in half, and the repair does not re-inflate the profile back to dense (Figure 5b). The retrained token embeddings rotate to a mean cosine of 0.32 with their pretrained selves at $2.26\times$ the norm. The repair is a global re-coding into the frozen quantized stack, not a restoration of the original network.

Two finer-grained pre-registered predictions failed, and we report them as nulls. The projected stack does not preserve a strong linearly-decodable carrier of the dense representation per layer (median per-layer linear-probe R^2 of 0.46, against the near-complete preservation the prediction required), and the repair is not legible at per-channel granularity: the change in each layer-norm gain correlates with neither that channel's quantization-induced bias (Spearman 0.10) nor its activation inflation (Spearman 0.08). All numbers in this appendix are a single evaluation at seed 0 and support the mechanism section qualitatively; the quantitative claims of Section 3.3 rest on the multi-seed cells of Figure 3.

J Pushing the frontier: what the survivor index buys

The $n{:}m$ index is the dominant term in the counted bit-width. A data-dependent magnitude mask must store it; a **fixed** mask reuses one shared pattern for every block, so its index cost is zero and the effective bit-width drops to the sign-plus-scale floor. Compared at *equal* $n{:}m$ and each at its own

subbit learning rate, the two keep the same active-weight count, sign bits, and forward-pass capacity; only survivor *selection* and the index accounting differ (Table 8).

Table 8: Fixed versus data-dependent (magnitude) mask at equal $n{:}m$ (enwik8, 0.86M anchor, 3 seeds).

mask	1:8	1:16
magnitude (data-dependent index)	0.59 bpw, 2.921 ± 0.046	0.41 bpw, 3.073 ± 0.060
fixed (shared pattern, $\sim$0 index)	0.219 bpw, 3.652 ± 0.020	**0.156** bpw, $\mathbf{3.721 \pm 0.006}$

The stored index buys about 0.6 to 0.7 bits per byte of survivor-selection quality (switching to a fixed mask costs $+0.73$ at 1:8 and $+0.65$ at 1:16), so indexing is not free; it purchases real quality, which is why counting it is correct. Yet even a fixed pattern at 0.156 bits per weight holds 3.72 bits per byte, far below naive post-hoc rounding at three times the bit budget and nowhere near random. A from-scratch model learns to route information through a fixed sparsity pattern, giving up the survivor-selection quality the stored index would have bought. This maps the native-QAT frontier to about 0.16 bits per weight.

K The token-matched scale sweep, arm by arm

Table 2 in the body reports the gap, which is what the claim is about. The two arms it is built from are here, so the shape of each can be read on its own rather than only through their difference. An earlier version of this sweep held the token budget fixed at 25M rather than matching tokens to parameters. We ran it, and we do not report it: at a fixed budget a larger model sees fewer tokens per parameter and is more undertrained [Ouyang et al., 2025], so any movement in the gap confounds scale with training adequacy. The token-matched curve below replaces it, and it is the only sweep we draw a scale claim from. Native QAT improves monotonically with scale at every width, which is what a working training path should do: at 1:16 it runs $2.973 \rightarrow 1.690$ bits per byte from 1.26M to 19.18M parameters, and the sub-1-bit arms stay between 1.4 and 3.0 throughout. Strong calibration PTQ does not: below one bit it sits at or above the order-0 byte-frequency floor of the enwik9 validation split (5.15 bpb) at every scale, and shows no systematic trend across it (the cells wobble by half a bit per byte in both directions, without tracking size), so part of the upward slope of the gap is PTQ pinned at that degenerate floor rather than a training-path cost that genuinely grows. This is why we read the positive slope as persistence and not as a widening law.

Table 9: The two arms behind Table 2, in validation bits per byte (enwik9, fixed 18 tokens per parameter, uniform L=6, three seeds per cell, each arm at its own bracketed best learning rate). The gap column of Table 2 is the difference of these. Every cell separates completely across its three seeds.

	binary ($\approx$ 1.0 bpw)		1:8		1:16	
scale	QAT	PTQ	QAT	PTQ	QAT	PTQ
1.26M	2.739	4.992	2.925	5.812	2.973	6.338
2.77M	2.005	4.829	2.323	6.076	2.472	6.795
4.87M	1.733	4.569	1.995	5.928	2.169	6.457
10.84M	1.540	4.266	1.699	5.230	1.850	6.086
19.18M	1.422	4.253	1.630	5.363	1.690	6.053

The reference solver at a second scale. The LB-ADMM arm of Table 1 is measured at the anchor, which leaves open whether the reference solver's failure is an artifact of a model too small for a low-rank factorization to have anything to work with. We therefore re-ran it at the 4.87M point of this sweep, $5.7\times$ the anchor, under the identical protocol: three seeds, its own default iteration budget, the calibration statistics of the calibration-PTQ arm, and rank set by the same accounting, which again lands on the budget rather than near it (0.358 against a 0.359 target at 1:16). It does not recover. At 1:16 LB-ADMM scores 7.137 ± 0.652 bits per byte against native QAT's 2.169 ± 0.010 at the same width, a gap of 4.97; at 1:8, 5.565 ± 0.209 against 1.995; and at binary, 3.677 ± 0.317 against 1.733. Every seed of every width separates from native QAT, and below one bit the reference

solver again sits above the order-0 byte-frequency floor of this split (5.15 bpb), where native QAT is far below it. We do not read the anchor and this point as a two-point trend: they are different corpora (enwik8 and enwik9), so their bits per byte are not on a common scale. The comparison that is clean is the one within this scale point, where all three solvers see the same data. There, LB-ADMM beats strong calibration PTQ at binary (3.677 against 4.569) and loses to it below one bit (7.137 against 6.457 at 1:16), so the reference solver is not uniformly the stronger post-hoc method; what both share is that neither comes near the training path. The gap is not a small-model artifact.

The mechanism replicates at a second configuration. Section 3.3 attributes the collapse to the missing re-solve, and its decisive arm is interface-only repair: freeze every quantized weight exactly as strong PTQ shipped it and retrain only the full-precision interface. That arm is measured at the anchor. We repeated it at the 4.87M point (1:16, full-pretraining adaptation budget, three seeds, at a learning rate bracketed within this configuration; the bracket is Table 10). It replicates. Holding every quantized bit fixed and retraining only the interface takes the projection from 6.761 ± 0.067 bits per byte to 2.668 ± 0.016, against native QAT's 2.169 ± 0.010 at the same width: 89% of the collapse recovered without re-solving a single quantized value. The re-solve is the dominant ingredient here as it is at the anchor.

The recovery is not complete, and it is less complete than at the anchor: a residual of 0.50 bits per byte separates repair from from-scratch QAT here, against 0.09 at the anchor. We do not read those two numbers as a trend. This configuration differs from the anchor in four ways at once, corpus (enwik9 against enwik8), width (256 against 128), depth (six layers against four), and adaptation budget (87.7M tokens against 25M), so nothing in this paper isolates which of them the larger residual belongs to, and we do not claim it is scale. We note only that the extra budget cannot be the explanation, since more adaptation should shrink the residual and this one is larger. What the second configuration does establish is that the mechanism is not an artifact of the anchor: the missing re-solve dominates the gap at both, and interface repair recovers most but not all of it.

Table 10: Interface-only repair at the 4.87M point (d=256, six layers, enwik9, 1:16, 87.7M adaptation tokens): the seed-0 learning-rate bracket. The first grid we ran (2e−3 to 3.2e−2) put its winner at the bottom edge, which is a bound rather than an optimum, so we extended downward before spending seeds. The minimum is interior but shallow: three rates lie within 0.013 bits per byte of each other.

repair LR	5e−4	1e−3	2e−3	8e−3	3.2e−2
repaired bpb	2.751	2.697	**2.685**	2.698	2.790

L Tokenization stress test: the gap survives a non-byte-level tokenization

The main experiments use byte-level enwik8, where the vocabulary is fixed by construction (256 byte values). This choice removes tokenizer effects, but it also raises the possibility that the result is tied to byte-level modeling. To check this, we repeat the anchor-scale comparison with a small learned BPE tokenizer (vocab 1000, enwik8, 2.2 bytes/token on average), the same 0.86M-parameter body, 3 seeds, per-arm bracketed learning rates, and the same disjoint calibration/validation split as the main anchor. Loss is converted to bits per byte by dividing by $\log 2$ and the measured bytes-per-token, so the numbers are on the same scale as Table 1.

Table 11: Tokenization stress test: BPE-tokenized enwik8 (vocab 1000, 0.86M body, 3 seeds, mean $\pm$ std bits per byte, each arm at its own bracketed best learning rate). The PTQ+fine-tune deficit column is a single seed-0 fine-tune at the 5M-token budget (n=1).

eff. bpw	native QAT	strong calib. PTQ	PTQ+finetune deficit
0.59 (1:8)	5.47 ± 0.06	9.60 ± 0.26	+0.22
0.41 (1:16)	5.53 ± 0.07	10.42 ± 0.44	not run

The gap direction reproduces on a tokenization where the model must also route information through learned subword units rather than raw bytes (Table 11). Native QAT beats calibration PTQ by +4.1 (1:8) and +4.9 (1:16) bits per byte, and the PTQ+fine-tune deficit at 1:8 (+0.22) is consistent with

the byte-level 5M-budget deficit ($+0.28$, Table 5). These BPE losses are used only for within-tokenization comparison; unlike the byte-level setup, the converted bpb is not bounded by the byte-uniform 8.0 reference when the token model is confidently wrong. This is a single BPE configuration at one scale, not a multi-corpus or multi-vocabulary sweep, so we read it as a direction check rather than a second error-barred anchor, and leave broader corpus coverage to future work.

M The chasm is not an enwik8 artifact: two further corpora

The main experiments are byte-level enwik8. To check that the separation is a property of the sub-1-bit regime rather than of one corpus, we repeat the anchor comparison on two further byte-level corpora that differ from enwik8 in opposite directions: the Python standard library (structured, highly repetitive, low entropy) and TinyStories [Eldan and Li, 2023] (simple natural-language narrative with a small effective vocabulary). The Python corpus is the `.py` files of the local CPython $3.11.7$ standard library, split at a file boundary so that no file appears in both halves, with a per-file manifest that makes it exactly reconstructible (Appendix B). Its training half is 11.9MB, so the 25M-token budget is about 2.1 passes over it, against under one pass for enwik8; every arm sees the identical data, so the within-corpus comparison is unaffected, but the absolute numbers are not comparable across corpora and we do not compare them.

Every arm re-brackets its own learning rate on each corpus at seed 0 before the three-seed runs, since an optimum does not transfer across data any more than it transfers across architecture. Both corpora select interior minima: on Python the native-QAT arm picks $1\mathrm{e}{-2}$ and full precision $2\mathrm{e}{-3}$, each with both neighbors worse, on TinyStories $2.5\mathrm{e}{-3}$, and in every case the grid was extended until the winner was no longer at an edge.

Table 12: The anchor comparison on two further corpora (0.86M anchor, byte-level, $1{:}16$ at 0.41 bpw, $n{=}3$, mean $\pm$ std bits per byte; each arm at its own bracketed interior rate). The gap is calibration PTQ minus native QAT. The final column is the order-0 byte-frequency floor of that corpus's own validation split, the score of a model that predicts marginal byte frequencies and nothing else. Enwik8 is shown for reference.

corpus	fp16	naive RTN	calibration PTQ	native QAT	order-0 floor
enwik8 (main)	2.449 ± 0.235	6.515 ± 0.184	6.447 ± 0.551	3.073 ± 0.060	5.06
Python stdlib	1.719 ± 0.006	6.190 ± 0.227	5.949 ± 0.399	2.622 ± 0.046	4.53
TinyStories	1.519 ± 0.174	6.185 ± 0.346	6.333 ± 0.311	2.230 ± 0.019	4.43

The separation reproduces on both, and it reproduces in the form that matters. Native QAT beats calibration PTQ by 3.33 bits per byte on Python and 4.10 on TinyStories, against 3.37 on enwik8, and on both corpora the per-seed separation is complete, so the same exact sign test applies. On both, one-shot post-hoc projection lands *past* the corpus's own order-0 byte-frequency floor, so below one bit it is beaten by predicting marginal byte frequencies, while native QAT stays well below that floor. That is the same statement the body makes on enwik8, on two corpora that differ from enwik8 and from each other.

Two things we do *not* claim. The gap is not systematically larger on the lower-entropy corpora: it is larger on TinyStories and, within seed noise, the same on Python as on enwik8, so these three points do not order the gap by corpus entropy and we draw no trend from them. And the enwik8 ordering of the two post-hoc arms is not stable: on TinyStories calibration PTQ scores slightly *worse* than naive RTN (6.33 against 6.19), on Python slightly better (5.95 against 6.19), and both differences sit inside the seed spread. The honest reading is that below one bit the two post-hoc arms are not reliably separable from each other, which is what the decoupling of the layer-local objective from the loss (Appendix G) would predict: once a better surrogate fit no longer buys lower loss, which of the two scores lower is not a property the method controls. Both are far above native QAT on every corpus, which is the claim the appendix exists to test.

N Deployment accounting in one place

The main text reports bits, bytes, quality, and latency where each is argued. Table 13 collects them for every method at the 0.41-bpw anchor, so the cost of each path can be read off a single row rather than assembled from four sections.

Table 13: Every path to the 0.41-bpw container at the 0.86M anchor (enwik8). Effective bpw counts the quantized block linears (survivor signs, $n{:}m$ index, per-row scale); model size is the whole serialized model, block linears at their counted width plus the full-precision interface that ships with them. Tokens are pretraining plus adaptation. The matmul column is the micro-benchmark ratio to a dense fp16 matmul on the deployed forward path, quoted over widths 128 to 512 (Appendix O); it is not an end-to-end decode measurement. Quality is mean $\pm$ std validation bits per byte, n as marked.

path	eff. bpw	model size	val bpb	tokens	matmul vs. dense
full precision (`fp16`)	16	1677 KiB	2.449 ± 0.235 (6)	25M	$1.00\times$
naive RTN	0.41	180 KiB	6.515 ± 0.184 (6)	25M	0.95 to $1.14\times$
strong calibration PTQ	0.41	180 KiB	6.447 ± 0.551 (6)	25M + 0.2M calib.	0.95 to $1.14\times$
binary factorization PTQ	0.40	180 KiB	6.308 ± 0.342 (6)	25M	1.26 to $1.31\times$
LB-ADMM [Chong et al., 2026]	0.40	180 KiB	6.470 ± 0.739 (6)	25M + 0.2M calib.	1.26 to $1.31\times$
PTQ + interface repair, 2%	0.41	180 KiB	4.153 ± 0.066 (3)	25M + 0.5M	0.95 to $1.14\times$
PTQ + QAT fine-tune, 20%	0.41	180 KiB	3.348 ± 0.050 (3)	25M + 5M	0.95 to $1.14\times$
PTQ + interface repair, 100%	0.41	180 KiB	3.165 ± 0.041 (3)	25M + 25M	0.95 to $1.14\times$
PTQ + QAT fine-tune, 100%	0.41	180 KiB	2.979 ± 0.033 (3)	25M + 25M	0.95 to $1.14\times$
native QAT from scratch	0.41	180 KiB	3.073 ± 0.060 (6)	25M	0.95 to $1.14\times$

Three things are visible in the table that no single section shows. First, every sub-1-bit row ships the same 180 KiB, so the column that separates them is quality, bought with tokens. Second, the whole-model compression is $9.3\times$ ($1677 \rightarrow 180$ KiB), not the $39\times$ the block linears alone would suggest: at this width the full-precision interface is 78% of the deployed bytes (Table 14), and its share falls as the model widens. Third, the cheapest row that reaches usable quality still costs a full training run of tokens, whether spent from scratch or after the projection.

Table 14: Whole-model size accounting at 1:16, at the anchor and at the largest scale point. The $39\times$ and $48\times$ figures are the block-linear reductions the effective bit-width refers to; the deployed model shrinks by less because the interface stays full precision.

	0.86M anchor (d=128)		19.18M (d=512)	
component	params	size	params	size
quantized block linears	786,432	39.0 KiB (from 1536)	18,874,368	774.0 KiB (from 36864)
full-precision interface	72,448	141.5 KiB	303,104	592.0 KiB
deployed model	858,880	180.5 KiB (from 1677.5)	19,177,472	1366.0 KiB (from 37456)
whole-model reduction		$\mathbf{9.3}\times$		$\mathbf{27.4}\times$
interface share of deployed bytes		78.4%		43.3%

Peak inference memory is a separate quantity and is not reduced here: the deployed path dequantizes each $n{:}m$ weight matrix to dense before the matmul, so activation and working memory match the dense model. The saving is in what is stored and in what must be read from memory to reconstruct the weights, which is what the storage column measures.

O Decode latency of the deployed sub-1-bit path

No fused sub-1-bit GEMM kernel exists on this backend, so the deployed forward pass dequantizes the $n{:}m$ weights to a dense matrix and runs a standard fp16 matmul. We time this with a square-matmul micro-benchmark (MLX, 200 timed iterations after warmup). At width 512 the dense baseline runs at $274\,\mu$s, the deployed dequantize-to-dense path at $261\,\mu$s (ratio 0.95), and a factored low-rank path at $345\,\mu$s (ratio 1.26); at width 128 the two ratios are 1.14 and 1.31. The deployed path is within noise of dense and the factored path is slower, so below one bit the regime buys storage, not decode speed, on this hardware.

Would a fast kernel fix this? The obvious reading of the paragraph above is that the missing piece is engineering: write a fused sub-1-bit kernel and the storage win becomes a speed win. We measured whether that is true, and on this hardware it is not. Using stock uniform quantization, for which MLX does ship fused kernels, we time single-token greedy decode on Qwen3-1.7B and Qwen3-4B across the weight-bit ladder $\{16, 6, 4, 3, 2\}$ at a fixed fp16 KV cache, context 2048, holding every weight variant resident and interleaving the configurations within each repetition so that shared thermal drift cancels in the per-repetition ratio (12 repetitions; fp16 baseline coefficient of variation 0.021 and 0.019 respectively). Speedups over fp16 are $1.39, 1.46, 1.51, 1.50$ for Qwen3-1.7B and $1.40, 1.89, 2.00, 2.23$ for Qwen3-4B at $6, 4, 3$ and 2 bits.

The two models behave differently, and the difference is the point. On Qwen3-1.7B the ladder is flat below 4 bits: halving the weight bytes from 4 bits to 2 bits buys 3%. On Qwen3-4B the same halving buys 18%, and the final step from 3 bits to 2 bits ($+0.23\times$) is larger than the step before it ($+0.11\times$) rather than smaller. Decode at batch size one does not saturate at 4B; it saturates at 1.7B. The reason is that single-token decode spends part of its time on work that does not shrink when weights shrink (KV traffic, activations, normalization, kernel dispatch, and dequantization arithmetic), and that part is a larger share of a smaller model. Fitting decode time as a bit-dependent weight term plus a bit-invariant remainder puts the remainder at 58% of decode on Qwen3-1.7B against 38% on Qwen3-4B. A larger share of decode is weight-bound at 4B than at 1.7B, so a larger share is in principle recoverable by cutting weight bits. Extrapolating the fits to zero bits suggests ceilings of $1.70\times$ and $2.64\times$ over fp16, against the $1.50\times$ and $2.23\times$ the shipped 2-bit kernel already reaches, but we report those as upper bounds only: the fit carries a bit-dependent weight term and a bit-invariant remainder, and no term for dequantization arithmetic growing as bits fall, which is precisely the term that binds below 2 bits. The measured saturation at 1.7B stands on its own without the fit. Context length does not change it: quadrupling context from 2048 to 8192 moves the 4-bit speedup on Qwen3-0.6B only from $1.14\times$ to $1.17\times$, so the bit-invariant remainder is not dominated by KV traffic.

Both halves of this are reported effects elsewhere, and the second bounds how far ours can be pushed. Malekar and Zand [2025] give the general form of the argument, that a fixed non-quantizable fraction of the workload ceilings any quantization speedup and that the fraction shrinks as models grow. Benazir and Lin [2025] profile decode on this same hardware family at 8B to 405B parameters and report the complementary effect at the other end of the bit ladder: below roughly 2 bits, dequantization arithmetic grows faster than the bandwidth it saves, the machine becomes bound by arithmetic rather than by memory, and a 2.6-bit model decodes faster than a 1.75-bit one on both Llama-70B and DeepSeek-236B. The trend we measure from 1.7B to 4B therefore must not be extrapolated past the sizes and bitwidths that produced it: on this hardware, the sub-2-bit regime is where a weight-only method stops paying, not where it starts. We claim only what we measure. At the sizes a laptop can train and serve at batch size one, sub-1-bit weights buy storage rather than latency, and the existing low-bit kernel measurements suggest limited remaining latency headroom on this hardware: the shipped 2-bit kernel already collects most of what cutting weight bits can give at these sizes. We did not implement a fused $n{:}m$ sub-1-bit kernel, so we do not evaluate one, and whether a sub-1-bit format could evade the wall is open. The mechanism Benazir and Lin [2025] identify is codebook lookup and unpacking bit widths that do not align to hardware lanes, and an $n{:}m$ mask requires neither, so we cannot rule out that a well-designed kernel does better than our extrapolation implies. Large-batch and prefill-dominated regimes are compute-bound rather than weight-bandwidth-bound and are likewise outside what we measure here. What the storage win does buy at batch size one is weight-loading bandwidth and disk residency, not activation memory: peak inference memory on the dequantize-to-dense path is dense-equivalent, because the dense matrix is materialized before the matmul.

P The precision tax is a real floor

Holding architecture and parameter count fixed and sweeping the bit-width gives the quality tax of quantization proper (Table 15).

Three error-barred observations. First, the one-bit tax is real but modest: the full-precision to binary gap (0.43 bits per byte) is about two pooled standard deviations, so quantization proper costs something even at one bit. Second, below one bit the tax does not order finely. Binary (1.09 bits) and $1{:}8$ (0.59 bits) are tied: their means differ by 0.04, well inside the binary arm's own run-to-run

Table 15: Precision tax (enwik8, 0.86M anchor, 6 runs, each arm at its own bracketed best learning rate).

bit-width	effective bpw	val bits/byte (mean $\pm$ std)	tax vs. fp16
`fp16`	16.0	2.452 ± 0.238	n/a
`binary`	1.09	2.879 ± 0.141	$+17.4\%$
`subbit 1:8`	0.59	2.921 ± 0.046	$+19.1\%$
`subbit 1:16`	0.41	3.073 ± 0.060	$+25.4\%$

spread (± 0.14), and their ± 1 standard-deviation bands overlap, so we make no claim that either is better. The one structural feature that survives noise is a step *up* at 0.41: 1:16 (3.073 ± 0.060) is worse than 1:8 (2.921 ± 0.046) by a non-overlapping 0.15, a 5.2% step. The conservative reading is therefore that binary and 1:8 tie, with a measurable cost at 0.41. Third, the tax is measured against a full-precision anchor at its own bracketed rate: the `fp16` sweep selects 2×10^{-3} by lowest three-seed mean (2.41) with both neighbors worse (1e−3: 2.49; 3e−3: 2.56; the seed-0 run alone dips lower at 3e−3, but its three-seed mean does not). The rate is selected on those three seeds; the anchor we report at that rate, 2.452 ± 0.238, is the mean over six. Table 1 reads 2.449 for the same cell because it pools the dense pretrains the PTQ harness itself trains, so that every post-hoc arm is paired to its own baseline by seed, and the disjoint replication of Appendix C reads 2.454. Three independent measurements of one quantity spanning 0.005 bits per byte sit an order of magnitude inside the run-to-run floor, and nothing in the tax turns on which of them we quote. The tax is a floor against a tuned baseline, not an artifact of an under-tuned one.

Q Extended related work

Sub-1-bit post-hoc compression. STBLLM [Dong et al., 2024] reaches 0.55 bits per weight by combining binary weights with an n:m structured mask, the same container class we train into natively. LittleBit [Lee et al., 2025] reaches 0.1 bits per weight through latent binary factorization, and NanoQuant [Chong et al., 2026] reaches binary and sub-1-bit rates with the LB-ADMM solver we run as a baseline in Section 3.1, wrapped in a multi-stage pipeline of staged block and model reconstruction and logit-level alignment. What separates these two is the budget of that optimization, not whether it happens. LittleBit learns its binary factors under quantization-aware training with distillation. NanoQuant never retrains end to end, and is post-training in that sense, but its block reconstruction still passes gradients through the binarized forward pass and its global scales are fitted to the full-precision model's logits, on 0.26M calibration tokens. Both optimize against the representation they deploy, which is the condition our decomposition isolates; what differs is how much of that optimization they need, and billion-parameter scale is where we would expect it to come cheapest. BTC-LLM [Gu et al., 2025] reaches 0.7 to 1.11 bits with codebooks and no sparse-mask overhead, learning its transformation by gradient descent on a calibration set.

STBLLM [Dong et al., 2024] is the one method here that meets neither description, and Section 4 treats it as the boundary case for our reading: a gradient-free one-shot projection that still holds a model together below one bit. What all of these routes share is a starting point, a pretrained full-precision model compressed downward. None asks what the same container yields when it is trained into from scratch, which is the comparison we run.

Native low-bit training. BitNet [Wang et al., 2023] and BitNet b1.58 [Ma et al., 2024] train low-bit models from scratch but floor at or above one bit per weight. The nearest work to ours, Sparse-BitNet [Zhang et al., 2026], combines 1.58-bit ternary weights with semi-structured 6:8 sparsity, but reports no sub-1-bit effective bit-width, so the sub-1-bit native regime is left open. We fix a magnitude-derived scale rather than learn it [Esser et al., 2020, Liu et al., 2020, 2022b], which keeps the bit accounting transparent at the cost of some quality.

How much training to spend against the representation. Beyond the ordering discussed in Section 4, two further results of Dremov et al. [2026] bear on ours. They find the loss-optimal share of compute spent in QAT grows with the total budget, and that this share is predicted by a tokens-per-parameter-byte statistic. Read at our bit-widths, that predicts the repair we are looking for should

get more expensive as the bit-width falls, not cheaper, which is what our budget curve shows (Appendix E).

Calibration PTQ. GPTQ [Frantar et al., 2022] and AWQ [Lin et al., 2024] reconstruct weight information from a Hessian or activation-aware surrogate. We run a strong baseline of this class, and Appendix G gives the reason it stops helping below one bit: the layer-local surrogate it optimizes decouples from the end-to-end loss, so a better surrogate fit stops predicting a better model. Methods that add auxiliary tuning on top of the projection [Li et al., 2023, Tseng et al., 2024, Egiazarian et al., 2024] buy fractions of a bit at 2 to 4 bits; the repair required at 0.41 bpw is qualitatively larger (Section 3.3).

Scaling behavior of quantization. ParetoQ [Liu et al., 2025] unifies 1 to 4 bit quantization-aware fine-tuning and locates a learning transition at or below two bits, consistent with the regime change we measure below one. Spectra [Kaushal et al., 2024] trains ternary models from scratch from 0.1B to 3.9B parameters. Both stop at or above one bit, and both operate far above our parameter range, so neither is a competitor to the measurement here so much as the boundary it extends past.

Kumar et al. [2025] model post-training quantization as an additive penalty (δ_{PTQ}) on the loss, and Ouyang et al. [2025] show across 160M to 12B checkpoints that post-hoc degradation grows both as models shrink and as they are trained on more tokens per parameter, which they read as a statement about how much a converged model has left to lose. Our anchor is the extreme small, fully-trained end of that trend, so the size of the collapse we measure is not a surprise to those laws.

Our scaling sweep does not neutralize their effect, and the persistence of the gap is not evidence against it. Ouyang et al. [2025] fit degradation in absolute size and tokens, $\Delta_q \propto D^{\beta}/N^{\alpha}$ with $\beta = 0.53$ and $\alpha = 0.23$, not in their ratio. Holding tokens per parameter fixed at 18 while width grows sends D up with N, and since $\beta > \alpha$ their law predicts the post-hoc penalty *grows* along exactly the curve we run. We take the direction only: our parameter range sits two orders of magnitude below their smallest checkpoint, and we do not extrapolate their fitted constants into it.

The dissociation is elsewhere and does not need their axis held fixed. Their Δ_q is the distance from a quantized model to its own full-precision parent, and their suite contains no model trained inside the container, so no term in their law speaks to the native arm, which occupies the same width, bit accounting and token budget as the post-hoc one and does not degrade at all. Their laws describe how far a projection falls; we measure that the fall is a property of projecting, not of the container, and that a model which never projects does not pay it.

R Broader impact

The result points toward paying training-scale optimization against the representation a small model will actually ship in, rather than rounding a trained model down and stopping. The upside is access: a model that ships at a ninth of its full-precision size is cheaper to store and distribute, which favors language models on personal and low-cost hardware, where inference stays local and data does not leave the device. The cost is training compute, paid whichever order the work comes in, so a lab that adopts this spends more energy than one that quantizes a model it has already trained; and our own evidence is that none of the cheaper routes we test reach usable quality at this scale. Efficient distribution also lowers the barrier to deploying models without the safeguards a served API can enforce, a general property of on-device compression rather than of this method. We study models under 20M parameters, with one single-seed cross-backend check at 43M, on public byte-level corpora, so nothing here constitutes a deployable capability; the practical consequences would arrive only if the separation survives at scales we do not test.