REPRESENTATION MATTERS IN 2-BIT QAT: FINITE BARNES–WALL REPRESENTATIONS FOR LARGE LANGUAGE MODELS

Anonymous authors
Paper under double-blind review

ABSTRACT

Quantization-aware training (QAT) is widely used to recover the quality of large language models (LLMs) at ultra-low precision, yet most existing methods focus on improving how a given low-bit representation is optimized. We study a complementary question: what representation should QAT optimize in the first place? We introduce BW-QAT, a representation-first framework for 2-bit weight-only QAT based on a finite 16-dimensional Barnes–Wall representation. BW-QAT encodes every 16 weights with a fixed 32-bit payload, calibrates the gain alphabet directly against the finite reconstruction space, and initializes trainable proxies from compensated pre-quantization targets. This initialization preserves the exact initial hard-quantized model while retaining the residual information needed for effective discrete updates. The resulting representation can be trained with hard quantized forward passes and a standard straight-through estimator. Across Qwen and Llama models, BW-QAT consistently outperforms matched lattice, scalar-QAT, and vector-QAT baselines in the low-data regime. On Qwen3-1.7B, it improves six-task average accuracy from 52.17% to 58.30% over a matched LC-QAT adaptation and reduces WikiText-2 perplexity from 18.90 to 14.02 across three training seeds. These results show that representation quality is a first-order factor in data-limited 2-bit QAT.

1 INTRODUCTION

Quantization is an effective way to reduce the memory and deployment cost of large language models (LLMs). The problem becomes much harder near two-bit precision, where a small discrete space can cause a large loss in model quality. Quantization-aware training (QAT) addresses this problem by optimizing the model under low-precision constraints. Recent methods improve QAT through more efficient training, better quantization functions, or more accurate gradient estimators (Chen et al., 2025; Liu et al., 2025b; Chen et al., 2026). This line of work has made optimization a central question in ultra-low-bit training.

We study a different but closely related question: *what discrete representation should QAT optimize?* QAT does not optimize an unconstrained continuous model. Each forward pass maps trainable variables into a fixed set of discrete states. At two bits, the choice of this set becomes critical. It determines the hard model available at the start of training and the reconstruction states that can be reached later. A weak representation can therefore create a poor starting point before optimization begins. This motivates the central view of our work: *representation quality is a first-order factor in data-limited 2-bit QAT*. At this precision, what QAT optimizes can matter as much as how it is optimized.

Structured vector representations offer more reconstruction choices under a fixed bit budget. Lattice and vector quantization have shown strong results in extreme post-training quantization and have recently been extended to QAT (Tseng et al., 2024; Kohli et al., 2026; Wang et al., 2026a). For QAT, however, a useful representation must do more than reconstruct the pretrained weights well. It should use a fixed rate, support exact hard decisions during training, and retain the same discrete form at deployment. It should also admit an optimization state from which the hard model can move effectively.

Reviewers: please read the Reviewer Guidelines (iclr.cc/Conferences/2027/ReviewerGuidelines) and the AI Policy for Reviewers (iclr.cc/Conferences/2027/AIPolicyForReviewers). If you used AI to expand, edit, or polish your review, please provide the input text to the LLM. Better still, consider skipping the LLM and submitting your original text: we, and the authors, are much more interested in your unedited thoughts than in what an LLM has to say. AI-assisted or not, you are putting your name and reputation behind your review: LLM-generated falsehoods, hallucinations or misrepresentations are subject to disciplinary action, which may include desk-rejecting all papers you have authored.

We introduce BW-QAT, a representation-first framework for 2-bit weight-only QAT based on a finite 16-dimensional Barnes–Wall representation. Each block of 16 weights is represented by a finite BW16 lattice point and a small gain index under a fixed 32-bit payload. BW-QAT calibrates the gain alphabet directly against the finite codec through *Codec-Consistent Gain Calibration* (CCGC), producing a strong initial hard model without changing the deployable representation.

The remaining challenge is trainability. Initializing the continuous proxy at the reconstructed codeword discards the residual information contained in the pre-quantization target and can leave the hard model nearly unchanged during QAT. We instead initialize the proxy as $Z_0 = Y_{\mathrm{comp}}$, where Y_{comp} is the compensated target that produced the initial hard decision. This preserves $Q_{\mathrm{BW}}(Z_0) = Q_0$ while retaining the continuous residual needed for discrete state changes. We refer to this design as *decision-preserving initialization*. After initialization, BW-QAT performs hard-forward QAT with the same finite BW representation that is later serialized for deployment.

Experiments across Qwen and Llama models support this representation-first view. Under the same 2-bit payload, BW16 consistently improves over matched Product-E_8 variants, and the resulting BW-QAT models also outperform matched scalar-QAT and vector-QAT baselines in the low-data regime. On Qwen3-1.7B, BW-QAT improves six-task average accuracy from 52.17% to 58.30% over the matched LC-QAT adaptation and reduces WikiText-2 perplexity from 18.90 to 14.02. Even when LC-QAT processes about $14\times$ more supervised labels under the same tested native-QAT time, its held-out NLL remains higher.

In summary, our main contributions are as follows:

- We propose BW-QAT, a fixed-rate 2-bit QAT framework built on a finite BW16 representation. CCGC calibrates the gain alphabet against the same finite codec used during training and deployment, allowing 16 weights to be represented with a fixed 32-bit payload.
- We introduce *decision-preserving initialization*, which initializes the continuous proxy from the compensated target while preserving the exact initial hard model. This separates the discrete model state from the continuous optimization state and retains the residual information needed for effective hard-state transitions.
- We provide controlled evidence that representation quality is a first-order factor in data-limited 2-bit QAT. Across matched lattice, scalar-QAT, and vector-QAT comparisons, BW-QAT achieves better low-data quality, and the advantage persists under matched native-QAT time.

2 Related Work

Ultra-low-bit quantization and structured representations. Low-bit post-training quantization (PTQ) has improved steadily through better reconstruction objectives, activation-aware calibration, outlier handling, and rotation. Representative methods include ZeroQuant, SmoothQuant, GPTQ, AWQ, OmniQuant, SpQR, SqueezeLLM, QuaRot, and SpinQuant (Yao et al., 2022; Xiao et al., 2023; Frantar et al., 2023; Lin et al., 2024; Shao et al., 2024; Dettmers et al., 2024; Kim et al., 2024; Ashkboos et al., 2024; Liu et al., 2025a). As the precision approaches two bits, several methods also enlarge the reconstruction space itself. QuIP combines incoherence processing with adaptive rounding, while QuIP# further introduces structured E_8 lattice codebooks (Chee et al., 2023; Tseng et al., 2024), and AQLM uses additive codebooks for extreme compression (Egiazarian et al., 2024). GPTVQ and VPTQ further show that higher-dimensional vector representations can improve the size–accuracy trade-off at very low precision (van Baalen et al., 2024; Liu et al., 2024a). UniSVQ parameterizes 2-bit codewords through affine transforms of integer lattices and provides a unified view of scalar and vector quantization (Wang et al., 2026b). These works make the structure and capacity of the discrete space an important design choice for extreme compression.

Quantization-aware training at extreme precision. QAT improves low-bit models by optimizing them while quantization is active in the forward pass. Existing methods reduce the difficulty of this process in different ways. LLM-QAT and BitDistiller use distillation to recover low-bit models, while EfficientQAT reduces the cost of full QAT (Liu et al., 2024b; Du et al., 2024; Chen et al., 2025). QA-LoRA, LR-QAT, and L4Q instead reduce the trainable state through low-rank adaptation (Xu et al., 2024; Bondarenko et al., 2024; Jeon et al., 2025). At extreme precision, the optimization

rule itself becomes increasingly important. PV-Tuning studies alternatives to standard STE-based updates, ParetoQ analyzes the training behavior of very low-bit models, StableQAT improves the backward surrogate, and Bit-by-Bit uses progressive precision reduction with outlier-channel splitting (Malinovskii et al., 2024; Liu et al., 2025b; Chen et al., 2026; Xu et al., 2026).

Structured vector representations have also entered QAT. Lattice-based QAT directly trains structured E_8 and D_4 projections with straight-through estimation (Kohli et al., 2026). LC-QAT introduces a linear-constrained vector representation that enables data-efficient 2-bit QAT with a differentiable parameterization (Wang et al., 2026a). These methods show that low-bit trainability depends on both the optimization procedure and the representation being optimized. BW-QAT studies a different point in this design space. It uses a finite fixed-rate BW16 representation, calibrates its gain alphabet against the finite codec, and preserves the same hard representation across initialization, QAT, and deployment. Decision-preserving initialization then separates the hard model state from the continuous optimization state while keeping the initial hard decision unchanged.

3 BW-QAT

3.1 Overview and Finite BW Representation

BW-QAT is a representation-first framework for 2-bit weight-only QAT. It represents each 16-weight block in a finite BW16 space under a fixed 32-bit payload and calibrates the gain alphabet against the same finite codec used by the quantizer. This initialization produces two coupled states: an initial hard model Q_0 and the compensated targets Y_{comp} that generate its discrete decisions. Decision-preserving initialization keeps Q_0 unchanged while using Y_{comp} as the continuous optimization state. Training then proceeds with hard BW16 states in the forward pass, and the final rank–gain records are directly serialized for deployment. Figure 1 illustrates how these components form the complete BW-QAT pipeline.

Finite BW16 representation. We first define the discrete space optimized by BW-QAT. Our representation starts from a 16-dimensional Barnes–Wall lattice and restricts it to a finite energy-shaped constellation, which enables high-dimensional block quantization under a fixed bit budget. We partition each target weight matrix into blocks of $d = 16$ weights. Let

$$D_{16} = \left\{ z \in \mathbb{Z}^{16} : \sum_{i=1}^{16} z_i \equiv 0 \pmod 2 \right\}, \tag{1}$$

and let $\mathrm{RM}(1,4) \subseteq \{0,1\}^{16}$ denote the first-order Reed–Muller code. We use the integer-scaled Barnes–Wall lattice

$$\Lambda_{\mathrm{BW}} = \bigcup_{c \in \mathrm{RM}(1,4)} (c + 2D_{16}). \tag{2}$$

To obtain a fixed-rate representation, we restrict the infinite lattice to the energy-shaped finite constellation

$$\mathcal{C}_T = \left\{ v \in \Lambda_{\mathrm{BW}} : \|v\|_2^2 \le T \right\}. \tag{3}$$

Our main 2-bit configuration uses $T = 36$, for which $|\mathcal{C}_{36}| = 237{,}307{,}681 < 2^{28}$. Each lattice point can therefore be represented by a 28-bit rank.

Each target matrix m is also associated with a positive 16-level gain alphabet $\Gamma_m = \{\gamma_{m,0}, \ldots, \gamma_{m,15}\}$. A quantized block is reconstructed as

$$q = \gamma_{m,j} v, \qquad v \in \mathcal{C}_{36}, \quad j \in \{0, \ldots, 15\}. \tag{4}$$

We store the lattice point with a 28-bit rank and the gain with a 4-bit index. Each 16-dimensional block therefore occupies exactly 32 payload bits, or 2 bits per target weight.

The large finite reconstruction space does not require an explicit codebook. For a continuous target $y \in \mathbb{R}^{16}$, we group candidate lattice points by their energy $E = \|v\|_2^2$ and define the best correlation on each reachable shell as

$$h_E(y) = \max_{\substack{v \in \Lambda_{\mathrm{BW}} \\ \|v\|_2^2 = E}} y^\top v. \tag{5}$$

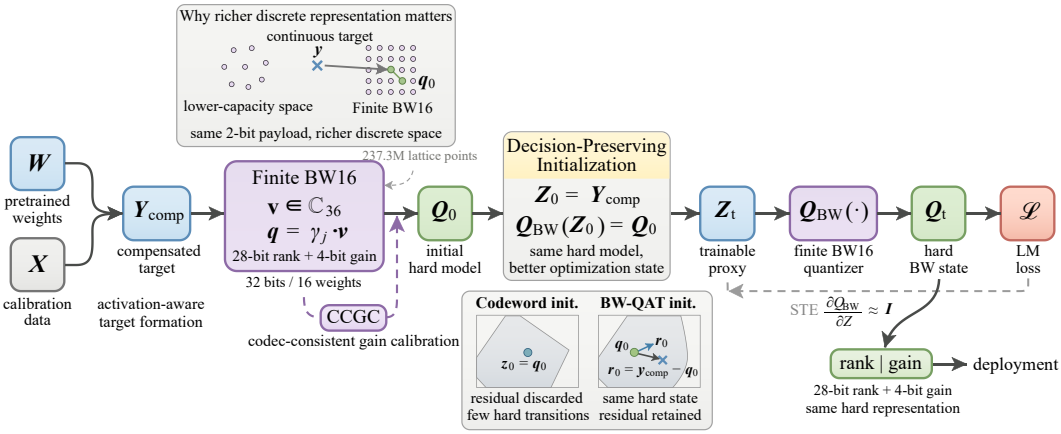


Figure 1: Overview of BW-QAT. The finite BW16 representation defines the initial hard model Q_0, while Y_{comp} is retained as the continuous optimization state. Decision-preserving initialization sets $Z_0 = Y_{\text{comp}}$ while preserving $Q_{\text{BW}}(Z_0) = Q_0$. Hard-forward QAT then uses the same finite representation during training and deployment.

For shell E and gain $\gamma_{m,j}$, the corresponding minimum squared distortion is

$$d_{E,j}(y) = \|y\|_2^2 - 2\gamma_{m,j} h_E(y) + \gamma_{m,j}^2 E. \tag{6}$$

The finite quantizer selects the shell and gain that minimize equation 6 and recovers the maximizing lattice point on that shell.

We compute the shell scores exactly by exploiting the Reed–Muller cosets, energy states, and parity constraints of BW16. The same structured state space supports completion counting and deterministic rank/unrank coding. This allows the codec to search and serialize the full finite constellation without materializing hundreds of millions of codewords. Appendix A gives the dynamic-programming recurrence, completion-counting procedure, and rank/unrank construction.

This finite codec defines the hard state space optimized by BW-QAT. We next describe how activation-aware compensation and codec-consistent gain calibration place the pretrained model into this space.

3.2 Codec-Consistent Initialization

A strong finite representation also requires a good placement of the pretrained weights within its discrete space. Two sources of structure are important for this initialization. The effect of a reconstruction error depends on the input activation statistics, while the lattice point and gain jointly determine each finite reconstruction. BW-QAT therefore combines activation-aware error compensation with gain calibration defined directly on the finite BW codec.

Activation-aware compensated targets. Consider a linear layer with weight matrix W and calibration activations X. We form the regularized empirical input metric

$$H_\delta = \frac{1}{N} X^\top X + \delta I. \tag{7}$$

We apply fixed orthogonal transforms R_o and R_i to the output and input coordinates,

$$\widetilde{W} = R_o W R_i^\top, \qquad \widetilde{H} = R_i H_\delta R_i^\top. \tag{8}$$

We then compute a block LDL factorization $\widetilde{H} = LDL^\top$ using 16-dimensional blocks.

We process each transformed weight row from right to left. When block k is processed, the blocks to its right have already been quantized. Rather than quantizing $\widetilde{w}_k$ directly, we form the compensated target

$$y_k = \widetilde{w}_k + \sum_{j>k} (\widetilde{w}_j - q_j) L_{jk}. \tag{9}$$

The correction term propagates the reconstruction errors of previously quantized blocks according to the activation-aware metric. The target y_k therefore accounts for earlier quantization decisions in the same row before it is mapped to the finite BW representation. We then apply the finite BW quantizer to y_k.

Codec-Consistent Gain Calibration. The lattice point and gain jointly determine every reconstruction in the finite representation. We therefore calibrate the gain alphabet with the same distortion used by the finite BW quantizer.

For a compensated target y_b, let $h_{b,E} = h_E(y_b)$. Using the shell distortion defined in equation 6, *Codec-Consistent Gain Calibration* (CCGC) solves

$$\min_{\Gamma_m} \sum_{b \in m} \min_{j,E} d_{E,j}(y_b). \tag{10}$$

We alternate between assigning each target to its best finite shell–gain pair and updating the gain values. For fixed assignments (E_b, j_b), each nonempty gain level has the closed-form update

$$\gamma_{m,j} = \frac{\sum_{b:j_b=j} h_{b,E_b}}{\sum_{b:j_b=j} E_b}. \tag{11}$$

CCGC thus calibrates the gain alphabet against the same finite decisions used by the BW codec. After calibration, we freeze the gain tables and rerun the sequential quantizer. This final pass returns the initial hard model Q_0 together with the compensated targets Y_{comp} that produced its discrete decisions. The former defines the step-zero hard model, while the latter will initialize the continuous QAT state in Sec. 3.3.

3.3 Decision-Preserving Initialization

A common QAT initialization sets the continuous proxy directly to the hard-quantized model, i.e., $Z_0 = Q_0$. For a low-bit vector quantizer, this choice removes the location of the continuous target that produced each hard decision. The proxy starts from the reconstructed codeword rather than from the original pre-quantization target. As a result, gradient updates can change the continuous proxy while leaving the hard state unchanged for many steps.

For a block-wise discrete representation, the hard state changes only when the proxy crosses a decision boundary and selects a different lattice point or gain. Initializing every proxy at its hard reconstruction therefore discards the residual that describes its position inside the current decision region.

BW-QAT instead reuses the compensated target produced during codec-consistent initialization. For each block b, the initial hard decision is obtained from y_b^{comp}. We initialize the trainable proxy with the same target,

$$z_b^{(0)} = y_b^{\text{comp}}, \qquad Q_{\text{BW}}\left(z_b^{(0)}\right) = q_b^{(0)}. \tag{12}$$

The step-zero hard state is therefore exactly the calibrated Q_0; only the continuous optimization state changes. The corresponding residual is

$$r_b^{(0)} = y_b^{\text{comp}} - q_b^{(0)}. \tag{13}$$

This residual preserves the position of the compensated target inside the current decision region. It gives optimization access to nearby hard states without changing the step-zero hard model.

We refer to this procedure as *decision-preserving initialization*. It separates the discrete model state from the continuous state used to optimize it. The former determines the hard model at step zero, while the latter determines where optimization begins. BW-QAT keeps the hard state fixed at initialization while retaining the pre-quantization residual in the continuous proxy.

3.4 Hard-Forward QAT and Deployment

After initialization, the finite constellation, gain tables, orthogonal transforms, and non-target parameters are fixed. The trainable state consists only of the continuous proxies Z associated with the target projection weights.

At optimization step t, each proxy block is quantized into the finite BW space,

$$Q_t = Q_{\mathrm{BW}}(Z_t), \tag{14}$$

where Q_t denotes the hard BW state. The effective layer weights are then reconstructed as

$$\widehat{W}_t = R_o^\top Q_t R_i. \tag{15}$$

Every forward pass therefore uses weights reconstructed from an actual finite BW state rather than from a separate soft representation.

We optimize the standard next-token cross-entropy objective. Because the finite BW quantizer is discrete, gradients are propagated to the continuous proxies with a straight-through estimator,

$$\frac{\partial Q_{\mathrm{BW}}(Z)}{\partial Z} \approx I. \tag{16}$$

After each optimizer update, the new proxy state is quantized again before the next forward pass. Crossing a decision boundary can change either the lattice rank or the gain index, which directly changes the hard state used in the following forward pass. No auxiliary training-time representation is introduced; the finite BW state is used throughout QAT.

At the end of training, the continuous proxies are discarded. Each final hard block is serialized by its 28-bit lattice rank and 4-bit gain index, i.e., $[\mathrm{rank}(v_b^\star),\ j_b^\star]$. The deployment package stores these finite BW records together with the matrix gain tables, fixed transforms, tensor metadata, and non-target parameters. The compensated targets, calibration statistics, and optimizer states are required only during initialization and training.

The nominal payload of the target weights remains exactly 2 bits per weight. To account for practical serialization overhead, we report both the realized target rate and the whole-model rate,

$$b_{\mathrm{target}} = \frac{8B_{\mathrm{target}}}{P_{\mathrm{target}}}, \qquad b_{\mathrm{whole}} = \frac{8(B_{\mathrm{target}} + B_{\text{non-target}})}{P_{\mathrm{all}}}. \tag{17}$$

All reported model-quality results are evaluated from independently reloaded deployment records, so the evaluated model corresponds to the same finite hard state represented by these serialized records.

4 Experiments

4.1 Experimental Setup

Models and protocols. We evaluate Qwen3-0.6B and Qwen3-1.7B (Yang et al., 2025) and Llama-3.2-1B (Meta AI, 2024). For BW-QAT, we quantize the q, k, v, o, gate, up, and down projections; embeddings, normalization layers, the language-model head, and other non-target parameters remain in high precision. Representation and ablation experiments use a frozen RedPajama CommonCrawl stream (Weber et al., 2024) with 64 calibration sequences of length 4096. Scalar-QAT experiments on Llama-3.2-1B use a 1:1 mixture of SlimPajama (Shen et al., 2023) and FineWeb-Edu (Penedo et al., 2024) at context length 2048, while vector-QAT experiments on Qwen3-1.7B use a fixed English shard of Ultra-FineWeb (Wang et al., 2025) at context length 4096. Training budgets count committed supervised labels; "1M" denotes exactly $1{,}048{,}576$ labels.

Baselines and evaluation. For representation controls, we compare BW16 with finite Product-E8 based on structured E_8 codebooks (Tseng et al., 2024) and a jointly shaped variant, JS-E8. All three use the same 32-bit payload per 16 weights and their own finite codec and sequential error-compensation path. For scalar QAT, we evaluate StableQAT (Chen et al., 2026), ParetoQ (Liu et al., 2025b), and DSQ (Gong et al., 2019) under the same Llama-3.2-1B model, target projections, training budget, and evaluation suite. For vector QAT, we use a matched LC-QAT adaptation (Wang et al., 2026a) on Qwen3-1.7B while preserving its quantization parameterization, trainable components, and training schedule. The main matched-label comparisons use three seeds; learning rates are selected on separate 100K-label development runs and then fixed. Matched-time comparisons use seed 0. We report held-out NLL, WikiText-2 (Merity et al., 2016) and C4 (Raffel et al., 2020) perplexity, and zero-shot accuracy with the LM Evaluation Harness (Biderman et al.,

2024). Avg8-S and Avg6-LC are the unweighted means over the eight- and six-task suites defined in Appendix C.4. Three-seed results are reported as mean ± sample standard deviation. All experiments run on Ubuntu on a server with an Intel Core Ultra 9 285K CPU, 256 GB RAM, and two NVIDIA RTX 5090 GPUs. Representation and scalar-QAT experiments use one GPU per run, while the Qwen3-1.7B vector-QAT experiments use two GPUs. Matched-time comparisons use the same device allocation within each comparison.

4.2 Does the Representation Matter?

We first isolate representation choice under a common 32-bit payload per 16 weights. Product-E8, JS-E8, and BW16 contain 83.2M, 182.0M, and 237.3M active points, respectively. Table 1 shows that the representation gap is already present before QAT begins. On Qwen3-0.6B, BW16 gives the lowest Q_0 NLL at 4.0133, compared with 4.6073 for JS-E8 and 4.9885 for Product-E8. The same ordering appears on Llama-3.2-1B.

Table 1: Finite-representation comparison under a common 32-bit payload per 16 weights. All values within each model use the same fixed held-out set. Llama-3.2-1B is evaluated through 100K supervised labels. Lower is better.

		Qwen3-0.6B			Llama-3.2-1B	
Representation	Active points	Q0 NLL ↓	100K ↓	1M ↓	Q0 NLL ↓	100K ↓
Product-E8	83.2M	4.9885	3.5294	3.2720	4.5756	3.2448
JS-E8	182.0M	4.6073	3.3954	3.1784	3.7064	3.1073
BW16	**237.3M**	**4.0133**	**3.3345**	**3.1388**	**3.5508**	**3.0455**

QAT reduces the separation between the representations but does not erase it. On Qwen3-0.6B, BW16 remains lowest at 1M labels with NLL 3.1388, compared with 3.1784 for JS-E8 and 3.2720 for Product-E8. Figure 2 shows the same trend over the training trajectory. The largest difference appears at Q_0, after which Product-E8 and JS-E8 recover progressively as they consume more training labels. JS-E8 also closes much of the gap to Product-E8, showing that a richer finite space can improve the low-data starting point. BW16 remains lower at every measured budget on both model families. Together, these trajectories indicate that representation choice primarily sets the initial hard model, while QAT determines how much of the initial gap is recovered during training.

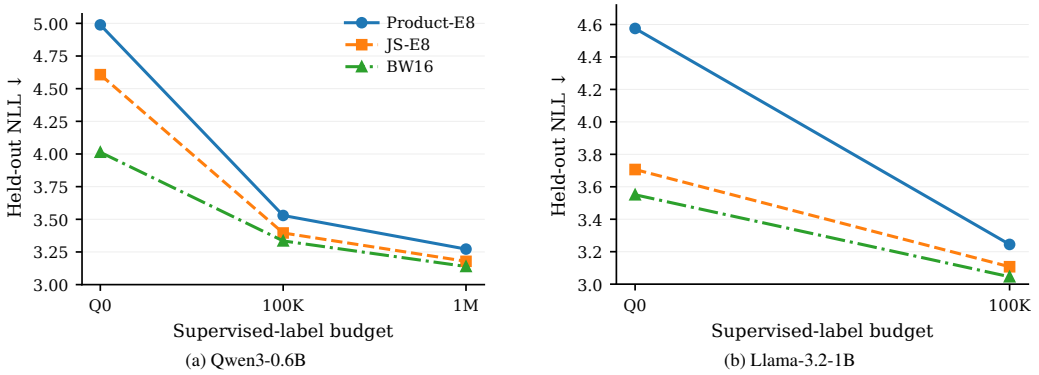


Figure 2: Effect of the finite representation on low-data QAT. Qwen3-0.6B shows the complete Q_0–100K–1M trajectory, while Llama-3.2-1B provides a cross-model comparison through 100K labels. BW16 remains lower in held-out NLL at every measured budget.

4.3 What Makes BW-QAT Trainable?

Table 2 separates two roles in the initialization. Comparing the two decision-preserving variants isolates the effect of CCGC: calibrating the gain alphabet against the finite codec lowers the Q_0 NLL from 4.3351 to 3.9668 and gives a modest additional improvement after 100K labels.

The two CCGC variants then isolate the effect of the continuous initialization. They start from exactly the same hard model, with Q_0 NLL $= 3.9668$. Codeword initialization produces only 5–9 cumulative hard-record changes over 100K supervised labels and leaves the NLL unchanged. Decision-preserving initialization instead produces roughly 359K–360K hard-record changes and reaches 3.2946 ± 0.0178 NLL. The difference therefore comes from the continuous optimization state rather than from a better step-zero hard model. CCGC improves where QAT starts; decision-preserving initialization makes that hard state mobile during training.

Table 2: Ablation of BW-QAT initialization on Qwen3-0.6B. The last column reports the range of cumulative hard-record changes over three seeds.

Variant	Q0 NLL ↓	100K NLL ↓	Hard changes
Fixed grid + decision-preserving	4.3351	3.3057 ± 0.0168	336K–342K
CCGC + codeword init.	**3.9668**	3.9668 ± 0.0000	5–9
CCGC + decision-preserving	**3.9668**	$\mathbf{3.2946 \pm 0.0178}$	**359K–360K**

4.4 Comparison with Modern 2-Bit QAT

We next compare BW-QAT with matched scalar- and vector-QAT baselines. These experiments examine the same representation effect from a different angle: whether the low-data advantage remains when alternative methods can process substantially more training data.

Scalar QAT. Table 3 compares all methods after the same $1{,}048{,}576$ supervised labels. BW-QAT reaches 2.8911 ± 0.0017 NLL and $51.90 \pm 0.05\%$ Avg8-S. Among the scalar baselines, ParetoQ gives the lowest NLL at 7.3971 ± 0.1149. The difference is already large at Q_0, consistent with the representation comparison above: the methods enter QAT from very different hard models.

Table 3: Matched 1M-label comparison on Llama-3.2-1B. Results are mean ± sample standard deviation over three training seeds. All methods use a 2-bit target payload.

Method	Q0 NLL ↓	1M NLL ↓	Avg8-S ↑
BW-QAT	**3.6334**	$\mathbf{2.8911 \pm 0.0017}$	$\mathbf{51.90 \pm 0.05}$
StableQAT	13.7030	7.9188 ± 0.1306	32.72 ± 0.26
ParetoQ	11.5957	7.3971 ± 0.1149	33.09 ± 0.20
DSQ	14.1729	8.1910 ± 0.1280	33.41 ± 0.31

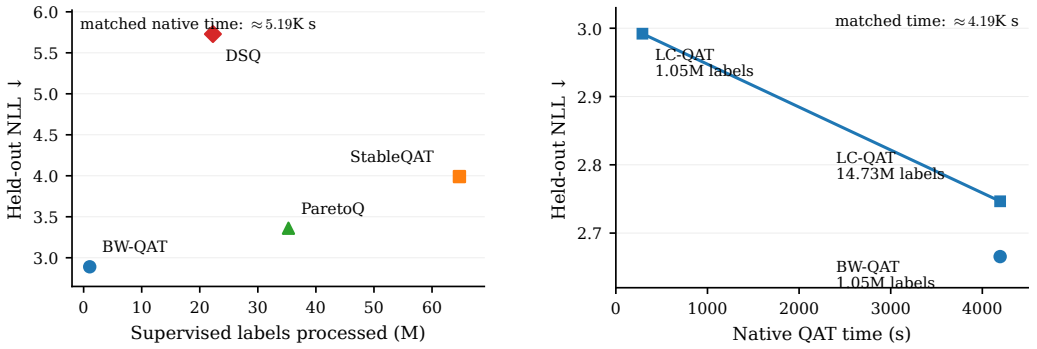
The scalar methods have much higher training throughput, so we also compare seed-0 endpoints under a common single-GPU wall-clock budget of about 5,190 seconds. As shown in Fig. 3(a), StableQAT, ParetoQ, and DSQ process 64.72M, 35.27M, and 22.27M labels, respectively, compared with 1.05M for BW-QAT. The additional data substantially improves the scalar baselines; ParetoQ, for example, reaches 3.360 NLL. Its endpoint nevertheless remains above the 2.890 NLL obtained by BW-QAT. The matched-time comparison therefore reveals two complementary effects: scalar QAT benefits from much higher throughput, whereas the BW representation provides substantially higher quality per supervised label.

Learned vector QAT. Table 4 compares BW-QAT with the matched LC-QAT adaptation at the same 1M-label budget on Qwen3-1.7B. Across three seeds, BW-QAT reaches 2.6663 ± 0.0006 NLL, compared with 2.9920 ± 0.0001 for LC-QAT. The same ordering appears on downstream metrics: BW-QAT obtains WikiText-2 perplexity 14.02 versus 18.90, C4 perplexity 18.77 versus 24.16, and Avg6-LC accuracy 58.30% versus 52.17%.

Figure 3(b) shows how the vector-QAT gap changes as LC-QAT uses its higher training throughput. LC-QAT reaches 2.9920 NLL after 1.05M labels in about 290 seconds and improves to 2.7466 after processing 14.73M labels within the common 4.19K-second budget. At the same wall-clock endpoint, BW-QAT reaches 2.6656 NLL after 1.05M labels. The gap therefore becomes substantially smaller as LC-QAT sees more data, but remains at the matched-time endpoint. This comparison separates the roles of representation and training throughput: BW-QAT provides the stronger low-data

Table 4: Three-seed 1M-label comparison on Qwen3-1.7B. The FP16 row gives the high-precision reference. Target and whole-model rates are realized serialized rates. The last column reports measured native QAT wall time for the 1M-label run.

Method	NLL ↓	WT2 ↓	C4 ↓	Avg6 ↑	Target / Whole bpw	Native QAT time (s)
FP16	2.3241	9.39	13.51	66.57	– / –	–
BW-QAT	$\mathbf{2.6663 \pm 0.0006}$	$\mathbf{14.02 \pm 0.01}$	$\mathbf{18.77 \pm 0.01}$	$\mathbf{58.30 \pm 0.43}$	2.029 / 4.610	4200.4
LC-QAT (matched)	2.9920 ± 0.0001	18.90 ± 0.00	24.16 ± 0.00	52.17 ± 0.06	**2.012 / 4.597**	**289.7**



(a) Scalar QAT under matched native time

(b) Native-time recovery against LC-QAT

Figure 3: Quality under matched native QAT time. (a) Llama-3.2-1B seed-0 endpoints after approximately 5,190 seconds; the horizontal axis shows the supervised labels processed by each method. (b) Qwen3-1.7B seed-0 LC-QAT endpoints at the 1M-label and matched-time budgets, together with the BW-QAT matched-time endpoint. Lines in panel (b) connect measured endpoints only.

starting point, while the faster LC-QAT trajectory progressively recovers with additional optimization.

4.5 Robustness and Practical Cost

Calibration size affects both initialization and short-QAT quality. Increasing the calibration set from 16 to 64 sequences lowers the Q_0 NLL from 4.2410 to 4.0133 and the 100K-label NLL from 3.3847 to 3.3345; the intermediate 32-sequence setting follows the same trend (4.1420 and 3.3608, respectively). Initialization time remains within 229–234 seconds across all three settings, and we use 64 sequences in the main experiments. The serialized representation remains close to the nominal 2-bit target payload. BW-QAT uses 2.006 target bpw and 5.041 whole-model bpw on Llama-3.2-1B, and 2.029 and 4.610 bpw on Qwen3-1.7B. The current exact BW implementation has higher per-step training cost: 81.10 s on Llama-3.2-1B and 65.52 s on Qwen3-1.7B, compared with 1.30–3.76 s for the scalar baselines and 4.47 s for LC-QAT. The matched-time experiments capture this trade-off directly: BW-QAT provides stronger quality per training label, while the faster baselines use their higher throughput to recover with additional data.

5 Conclusion

We studied 2-bit QAT from a representation-first perspective and introduced BW-QAT, which combines a fixed-rate finite BW16 representation with codec-consistent gain calibration and decision-preserving initialization. The finite representation provides a strong initial hard model, while the compensated targets retain a continuous optimization state without changing the step-zero hard decisions. Hard-forward QAT then optimizes the same discrete representation that is serialized for deployment. Across controlled lattice, scalar-QAT, and vector-QAT comparisons on Qwen and Llama models, BW-QAT consistently improves low-data model quality, and its advantage remains under matched native-QAT time. These results show that ultra-low-bit QAT is not only an optimization problem: the discrete representation itself is a first-order design choice.

AI USE STATEMENT

We used generative AI tools to assist with language editing and polishing, literature retrieval and identification of related work, and code revision. All AI-assisted outputs were reviewed by the authors. We take full responsibility for the final content of this work.

ETHICS STATEMENT

This work studies model quantization and quantization-aware training for large language models. The experiments use existing pretrained model checkpoints and established text datasets, and do not involve human-subject studies or the collection of new personal data. The proposed method is intended to improve the efficiency of low-precision model deployment. We are not aware of ethical risks specific to BW-QAT beyond those generally associated with the underlying pretrained language models and datasets.

REPRODUCIBILITY STATEMENT

We provide the algorithmic and experimental details needed to reproduce the reported results. The main paper specifies the finite BW16 representation, codec-consistent gain calibration, decision-preserving initialization, and hard-forward QAT procedure. Appendices A and B provide the exact finite-codec construction and additional initialization and training details. Appendix C reports model revisions, data protocols, optimization settings, baseline matching, evaluation procedures, and timing definitions, while Appendix D provides per-seed results and additional experimental details. Anonymous source code and configuration files are included in the supplementary material.

REFERENCES

Saleh Ashkboos, Amirkeivan Mohtashami, Maximilian L. Croci, Bo Li, Pashmina Cameron, Martin Jaggi, Dan Alistarh, Torsten Hoefler, and James Hensman. Quarot: Outlier-free 4-bit inference in rotated llms. In *Advances in Neural Information Processing Systems*, volume 37, 2024. doi: 10.52202/079017-3180.

Stella Biderman, Hailey Schoelkopf, Lintang Sutawika, Leo Gao, Jonathan Tow, et al. Lessons from the trenches on reproducible evaluation of language models. *arXiv preprint arXiv:2405.14782*, 2024.

Yelysei Bondarenko, Riccardo Del Chiaro, and Markus Nagel. Low-rank quantization-aware training for llms. *arXiv preprint arXiv:2406.06385*, 2024.

Jerry Chee, Yaohui Cai, Volodymyr Kuleshov, and Christopher M. De Sa. Quip: 2-bit quantization of large language models with guarantees. In *Advances in Neural Information Processing Systems*, volume 36, 2023. doi: 10.52202/075280-0196.

Mengzhao Chen, Wenqi Shao, Peng Xu, Jiahao Wang, Peng Gao, Kaipeng Zhang, and Ping Luo. EfficientQAT: Efficient quantization-aware training for large language models. In *Proceedings of the 63rd Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pp. 10081–10100. Association for Computational Linguistics, 2025. doi: 10.18653/v1/2025.acl-long.498.

Tianyi Chen, Sihan Chen, Xiaoyi Qu, Dan Zhao, Ruomei Yan, Jongwoo Ko, Luming Liang, and Pashmina Cameron. StableQAT: Stable quantization-aware training at ultra-low bitwidths. *arXiv preprint arXiv:2601.19320*, 2026.

Tim Dettmers, Ruslan Svirschevski, Vage Egiazarian, Denis Kuznedelev, Elias Frantar, Saleh Ashkboos, Alexander Borzunov, Torsten Hoefler, and Dan Alistarh. Spqr: A sparse-quantized representation for near-lossless llm weight compression. In *International Conference on Learning Representations*, 2024.

DaYou Du, Yijia Zhang, Shijie Cao, Jiaqi Guo, Ting Cao, Xiaowen Chu, and Ningyi Xu. Bitdistiller: Unleashing the potential of sub-4-bit llms via self-distillation. In *Proceedings of the 62nd*

Annual Meeting of the Association for Computational Linguistics, pp. 102–116. Association for Computational Linguistics, 2024. doi: 10.18653/v1/2024.acl-long.7.

Vage Egiazarian, Andrei Panferov, Denis Kuznedelev, Elias Frantar, Artem Babenko, and Dan Alistarh. Extreme compression of large language models via additive quantization. In *Proceedings of the 41st International Conference on Machine Learning*, volume 235 of *Proceedings of Machine Learning Research*, pp. 12284–12303. PMLR, 2024.

Elias Frantar, Saleh Ashkboos, Torsten Hoefler, and Dan Alistarh. Gptq: Accurate post-training quantization for generative pre-trained transformers. In *International Conference on Learning Representations*, 2023.

Ruihao Gong, Xianglong Liu, Shenghu Jiang, Tianxiang Li, Peng Hu, Jiazhen Lin, Fengwei Yu, and Junjie Yan. Differentiable soft quantization: Bridging full-precision and low-bit neural networks. In *Proceedings of the IEEE/CVF International Conference on Computer Vision*, 2019.

Hyesung Jeon, Yulhwa Kim, and Jae-Joon Kim. L4q: Parameter efficient quantization-aware fine-tuning on large language models. In *Proceedings of the 63rd Annual Meeting of the Association for Computational Linguistics*, pp. 2002–2024. Association for Computational Linguistics, 2025. doi: 10.18653/v1/2025.acl-long.99.

Sehoon Kim, Coleman Richard Charles Hooper, Amir Gholami, Zhen Dong, Xiuyu Li, Sheng Shen, Michael W. Mahoney, and Kurt Keutzer. Squeezellm: Dense-and-sparse quantization. In *Proceedings of the 41st International Conference on Machine Learning*, volume 235 of *Proceedings of Machine Learning Research*, pp. 23901–23923. PMLR, 2024.

Rishika Kohli, Soma S Dhavala, Shaifu Gupta, and Manoj Singh Gaur. Lattice-based vector quantization for low-bit quantization-aware training. In *Conference on Parsimony and Learning*, volume 328 of *Proceedings of Machine Learning Research*, pp. 218–241. PMLR, 2026.

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

Yifei Liu, Jicheng Wen, Yang Wang, Shengyu Ye, Li Lyna Zhang, Ting Cao, Cheng Li, and Mao Yang. Vptq: Extreme low-bit vector post-training quantization for large language models. In *Proceedings of the 2024 Conference on Empirical Methods in Natural Language Processing*. Association for Computational Linguistics, 2024a.

Zechun Liu, Barlas Oguz, Changsheng Zhao, Ernie Chang, Pierre Stock, Yashar Mehdad, Yangyang Shi, Raghuraman Krishnamoorthi, and Vikas Chandra. Llm-qat: Data-free quantization aware training for large language models. In *Findings of the Association for Computational Linguistics: ACL 2024*, pp. 467–484. Association for Computational Linguistics, 2024b. doi: 10.18653/v1/2024.findings-acl.26.

Zechun Liu, Changsheng Zhao, Igor Fedorov, Bilge Soran, Dhruv Choudhary, Raghuraman Krishnamoorthi, Vikas Chandra, Yuandong Tian, and Tijmen Blankevoort. Spinquant: Llm quantization with learned rotations. In *International Conference on Learning Representations*, 2025a.

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. In *Advances in Neural Information Processing Systems*, volume 38, 2025b. doi: 10.52202/085713-3055.

Vladimir Malinovskii, Denis Mazur, Ivan Ilin, Denis Kuznedelev, Konstantin Burlachenko, Kai Yi, Dan Alistarh, and Peter Richtarik. Pv-tuning: Beyond straight-through estimation for extreme llm compression. In *Advances in Neural Information Processing Systems*, volume 37, 2024. doi: 10.52202/079017-0165.

Stephen Merity, Caiming Xiong, James Bradbury, and Richard Socher. Pointer sentinel mixture models. *arXiv preprint arXiv:1609.07843*, 2016.

Meta AI. Llama 3.2: Revolutionizing edge ai and vision with open, customizable models. Meta AI Blog, 2024. Released September 25, 2024.

Guilherme Penedo, Hynek Kydlíček, Loubna Ben allal, Anton Lozhkov, Margaret Mitchell, Colin Raffel, Leandro von Werra, and Thomas Wolf. The fineweb datasets: Decanting the web for the finest text data at scale. *arXiv preprint arXiv:2406.17557*, 2024.

Colin Raffel, Noam Shazeer, Adam Roberts, Katherine Lee, Sharan Narang, Michael Matena, Yanqi Zhou, Wei Li, and Peter J. Liu. Exploring the limits of transfer learning with a unified text-to-text transformer. *Journal of Machine Learning Research*, 21(140):1–67, 2020.

Wenqi Shao, Mengzhao Chen, Zhaoyang Zhang, Peng Xu, Lirui Zhao, Zhiqian Li, Kaipeng Zhang, Peng Gao, Yu Qiao, and Ping Luo. Omniquant: Omnidirectionally calibrated quantization for large language models. In *International Conference on Learning Representations*, 2024.

Zhiqiang Shen, Tianhua Tao, Liqun Ma, Willie Neiswanger, Zhengzhong Liu, Hongyi Wang, Bowen Tan, Joel Hestness, Natalia Vassilieva, Daria Soboleva, and Eric Xing. Slimpajama-dc: Understanding data combinations for llm training. *arXiv preprint arXiv:2309.10818*, 2023.

Albert Tseng, Jerry Chee, Qingyao Sun, Volodymyr Kuleshov, and Christopher De Sa. QuIP#: Even better llm quantization with hadamard incoherence and lattice codebooks. In *Proceedings of the 41st International Conference on Machine Learning*, volume 235 of *Proceedings of Machine Learning Research*, pp. 48630–48656. PMLR, 2024.

Mart van Baalen, Andrey Kuzmin, Markus Nagel, Peter Couperus, Cedric Bastoul, Eric Mahurin, Tijmen Blankevoort, and Paul Whatmough. Gptvq: The blessing of dimensionality for llm quantization. *arXiv preprint arXiv:2402.15319*, 2024.

Haoyu Wang, Xingyu Yu, Haiyan Zhao, Fengxiang Wang, and Xu Han. LC-QAT: Data-efficient 2-bit qat for llms via linear-constrained vector quantization. *arXiv preprint arXiv:2606.10531*, 2026a.

Haoyu Wang, Haiyan Zhao, Xingyu Yu, Zhangyang Yao, Xu Han, Zhiyuan Liu, and Maosong Sun. Unisvq: 2-bit unified scalar-vector quantization. *arXiv preprint arXiv:2606.10520*, 2026b.

Yudong Wang, Zixuan Fu, Jie Cai, Peijun Tang, Hongya Lyu, Yewei Fang, Zhi Zheng, Jie Zhou, Guoyang Zeng, Chaojun Xiao, Xu Han, and Zhiyuan Liu. Ultra-fineweb: Efficient data filtering and verification for high-quality llm training data. *arXiv preprint arXiv:2505.05427*, 2025.

Maurice Weber, Daniel Fu, Quentin Anthony, Yonatan Oren, Shane Adams, Anton Alexandrov, Xiaozhong Lyu, Huu Nguyen, Xiaozhe Yao, Virginia Adams, Ben Athiwaratkun, Rahul Chalamala, Kezhen Chen, Max Ryabinin, Tri Dao, Percy Liang, Christopher Ré, Irina Rish, and Ce Zhang. Redpajama: An open dataset for training large language models. *arXiv preprint arXiv:2411.12372*, 2024.

Guangxuan Xiao, Ji Lin, Mickael Seznec, Hao Wu, Julien Demouth, and Song Han. Smoothquant: Accurate and efficient post-training quantization for large language models. In *Proceedings of the 40th International Conference on Machine Learning*, volume 202 of *Proceedings of Machine Learning Research*, pp. 38087–38099. PMLR, 2023.

Binxing Xu, Hao Gu, Lujun Li, Hao Wang, Bei Liu, Jiacheng Liu, Qiyuan Zhu, Xintong Yang, Chao Li, Sirui Han, and Yike Guo. Bit-by-bit: Progressive qat strategy with outlier channel splitting for stable low-bit llms. In *Proceedings of the 64th Annual Meeting of the Association for Computational Linguistics*, pp. 22284–22299. Association for Computational Linguistics, 2026. doi: 10.18653/v1/2026.acl-long.1020.

Yuhui Xu, Lingxi Xie, Xiaotao Gu, Xin Chen, Heng Chang, Hengheng Zhang, Zhengsu Chen, Xiaopeng Zhang, and Qi Tian. Qa-lora: Quantization-aware low-rank adaptation of large language models. In *International Conference on Learning Representations*, 2024.

An Yang et al. Qwen3 technical report. *arXiv preprint arXiv:2505.09388*, 2025.

Zhewei Yao, Reza Yazdani Aminabadi, Minjia Zhang, Xiaoxia Wu, Conglong Li, and Yuxiong He. Zeroquant: Efficient and affordable post-training quantization for large-scale transformers. In *Advances in Neural Information Processing Systems*, volume 35, 2022. doi: 10.52202/068431-1970.

A EXACT FINITE BW16 CODEC

The main paper defines the finite BW16 representation and its fixed-width 32-bit record. This section gives the exact search and coding procedure underlying that representation. The key point is that the quantizer can search the finite constellation implicitly, without materializing all 237,307,681 points in $\mathcal{C}_{36}$.

A.1 EXACT FINITE SEARCH

For each Reed–Muller coset $c \in \mathrm{RM}(1,4)$, we solve the finite nearest-neighbor problem with a dynamic program over the BW16 constraints. Let

$$F_c(i, e, p) \tag{18}$$

denote the maximum partial correlation after processing the first i coordinates, where e is the accumulated squared energy and p is the parity state. The initial state is

$$F_c(0, 0, 0) = 0, \tag{19}$$

while all other initial states have value $-\infty$.

At coordinate $i+1$, a candidate integer a must satisfy $a \equiv c_{i+1} \pmod 2$. Define

$$\pi_c(a) = \left(\frac{a - c_{i+1}}{2}\right) \bmod 2. \tag{20}$$

The dynamic-programming transition is

$$F_c\big(i+1,\, e + a^2,\, p \oplus \pi_c(a)\big) = \max\left\{F_c(i, e, p) + y_{i+1}a\right\}, \tag{21}$$

where the maximization is taken over feasible predecessor states that remain within the finite energy bound. A valid terminal state has $i = 16$ and parity $p = 0$. The best correlation on a reachable energy shell E is therefore

$$h_E(y) = \max_{c \in \mathrm{RM}(1,4)} F_c(16, E, 0). \tag{22}$$

Given a frozen gain alphabet Γ_m, the finite BW quantizer selects the gain and shell that minimize

$$\gamma_{m,j}^2 E - 2\gamma_{m,j} h_E(y), \tag{23}$$

because the omitted term $\|y\|_2^2$ is constant across candidates. Backtracking through the selected dynamic-programming states then recovers the corresponding lattice point.

The search is deterministic. Ties inside the dynamic program are resolved by the first maximizing predecessor under a fixed incoming-edge order. The final quantization decision uses a fixed gain-index–energy–coset ordering. These rules are shared by initialization, QAT, and deployment.

A.2 COMPLETION COUNTING AND RANK/UNRANK

The same structured state space also provides a fixed-width representation of the selected lattice point. Every valid point in $\mathcal{C}_{36}$ is assigned a unique integer rank using exact suffix-completion counts over the coordinate–energy–parity states.

Candidate points follow a fixed Reed–Muller coset order and, within each coset, a signed-coordinate lexicographic order. To rank a point, the encoder traverses its coordinates and accumulates the completion counts of feasible branches that precede the observed branch. Unranking performs the inverse operation by selecting the unique branch whose completion-count interval contains the requested rank.

The valid rank range is

$$0, \ldots, 237{,}307{,}680, \tag{24}$$

which fits in 28 bits; codes outside this range are rejected. Together with the 4-bit gain index, this gives the 32-bit record used throughout the paper.

This construction makes search, ranking, and decoding deterministic while avoiding a dense codebook. Existing codec checks cover the canonical test suite, the release test suite, and 65,536 real projection blocks, including rank/unrank and reconstruction round trips.

B Initialization and Hard-Forward QAT Details

The main paper presents the initialization at the level needed to explain BW-QAT. Here we provide the implementation details that connect activation-aware compensation, CCGC, decision-preserving initialization, and hard-forward QAT.

B.1 Activation-Aware Compensation

For calibration activations X, let

$$G = \frac{1}{N} X^\top X. \tag{25}$$

The implementation evaluates the regularized input metric in normalized coordinates:

$$\bar{H}_\delta = \frac{G}{\mathrm{mean}(\mathrm{diag}(G))} + 0.01 I. \tag{26}$$

This is the normalized implementation of the metric in Sec. 3.2. Equivalently,

$$\bar{H}_\delta = \frac{1}{\mu}\left(G + 0.01\mu I\right), \qquad \mu = \mathrm{mean}(\mathrm{diag}(G)). \tag{27}$$

The positive global scaling changes the block-diagonal factor D but leaves the unit-lower feedback factor L unchanged, so the compensated-target construction is consistent with the formulation in the main text.

We apply the fixed transforms as

$$\widetilde{H} = R_i \bar{H}_\delta R_i^\top, \qquad \widetilde{W} = R_o W R_i^\top. \tag{28}$$

The transforms R_o and R_i are deterministic and remain fixed during QAT.

We then compute a block LDL factorization

$$\widetilde{H} = L D L^\top \tag{29}$$

with block size 16. Each transformed weight row is processed from right to left. When block k is quantized, the blocks $j > k$ have already been assigned hard BW states, and the compensated target is

$$y_k = \widetilde{w}_k + \sum_{j>k} (\widetilde{w}_j - q_j) L_{jk}. \tag{30}$$

This target is passed to the finite BW quantizer defined in Appendix A.

B.2 Codec-Consistent Gain Calibration

The lattice point and gain jointly define each finite reconstruction. CCGC therefore calibrates the gain alphabet using the same finite-codec distortion that is used to make the hard quantization decision.

Each target matrix uses a positive 16-level gain alphabet. A provisional sequential pass first records the compensated targets produced by the current codec. The gain values are initialized from log-spaced quantiles of the target radii and are then fitted to these compensated targets.

Using the shell distortion $d_{E,j}(y)$ defined in the main text, CCGC solves

$$\min_{\Gamma_m} \sum_{b \in m} \min_{j,E} d_{E,j}(y_b). \tag{31}$$

For fixed shell–gain assignments (E_b, j_b), a nonempty gain level is updated as

$$\gamma_{m,j} = \frac{\sum_{b:j_b=j} h_{E_b}(y_b)}{\sum_{b:j_b=j} E_b}. \tag{32}$$

If a gain level receives no valid assignment, its previous value is retained. The positive gains are sorted after every update.

Calibration runs for at most 20 iterations and terminates earlier when the relative objective improvement falls below 10^{-7}. After calibration, the gain table is frozen and the full sequential quantization pass is repeated. This final pass produces the hard model Q_0 together with the compensated targets Y_{comp} that generate its final discrete decisions.

B.3 Decision-Preserving Initialization

The final sequential pass satisfies, for every target block,

$$q_b^{(0)} = Q_{\mathrm{BW}}\left(y_b^{\mathrm{comp}}\right). \tag{33}$$

BW-QAT initializes the corresponding continuous proxy as

$$z_b^{(0)} = y_b^{\mathrm{comp}}. \tag{34}$$

Proposition 1. Let Q_{BW} be deterministic under fixed gains and fixed tie-breaking. If $q_b^{(0)} = Q_{\mathrm{BW}}(y_b^{\mathrm{comp}})$ and $z_b^{(0)} = y_b^{\mathrm{comp}}$, then

$$Q_{\mathrm{BW}}\left(z_b^{(0)}\right) = q_b^{(0)}. \tag{35}$$

Proof. Both expressions evaluate the same deterministic quantizer on the same input. □

The continuous proxy nevertheless retains the residual

$$r_b^{(0)} = y_b^{\mathrm{comp}} - q_b^{(0)}. \tag{36}$$

Thus, the initialization preserves the step-zero hard state while retaining the pre-quantization offset in the continuous optimization state.

B.4 Hard-Forward Training and Deployment

After initialization, the finite constellation, gain tables, transforms, and non-target model parameters remain fixed. The target projection matrices are represented by continuous FP32 proxies.

At optimizer step t,

$$Q_t = Q_{\mathrm{BW}}(Z_t), \tag{37}$$

and the forward pass reconstructs the effective weights from this hard BW state. Gradients are propagated to the proxies with the identity straight-through estimator,

$$\frac{\partial Q_{\mathrm{BW}}(Z)}{\partial Z} \approx I. \tag{38}$$

The hard state is refreshed after every optimizer update.

After training, the continuous proxies and optimizer state are discarded. Each target block is stored using its 28-bit BW rank and 4-bit gain index, together with the frozen gain tables, transform metadata, tensor metadata, and retained non-target tensors. The reported deployment evaluations independently reload these packed hard records rather than the continuous training proxies.

C Experimental Protocol

The main paper keeps the experimental setup concise. This section records the model revisions, optimization settings, evaluation protocol, and comparison conditions used to obtain the reported results.

C.1 Models and Target Scope

Table 5: Models used in the controlled experiments. The complete model and tokenizer revisions are reported for reproducibility.

Model	Revision	Total params.	Target params.
Qwen3-0.6B	`da87bfb608c14b7cf20ba1ce41287e8de496c0cd`	596.0M	440.4M
Llama-3.2-1B	`4e20de362430cd3b72f300e6b0f18e50e7166e08`	1.236B	973.1M
Qwen3-1.7B	`ea980cb0a6c2ae4b936e82123acc929f1cec04c1`	1.721B	1.409B

For BW-QAT, the target matrices are the q, k, v, o, gate, up, and down projections. Embeddings, normalization layers, the language-model head, and other non-target parameters remain in high precision. Baseline-specific trainable states are summarized in Sec. C.5.

C.2 Training and Calibration Protocols

The experiments use three controlled tracks. Representation experiments isolate the discrete representation, initialization ablations isolate the role of the QAT proxy state, and the scalar/vector tracks compare BW-QAT with modern QAT methods.

Table 6: Training protocols used in the controlled comparisons. "Labels/span" denotes supervised next-token targets rather than the raw number of input tokens.

Track	Model	Training data	Labels / span	Calibration	Seeds
Representation	Qwen3-0.6B, Llama-3.2-1B	RedPajama CommonCrawl	2048	64×4096	0
Initialization ablation	Qwen3-0.6B	RedPajama CommonCrawl	2048	64×4096	0,1,2
Scalar QAT	Llama-3.2-1B	SlimPajama + FineWeb-Edu	2048	BW: 64×4096	0,1,2
Vector QAT	Qwen3-1.7B	Ultra-FineWeb	4096	shared 1024×4096	0,1,2

A committed supervised label is a valid next-token target belonging to an optimizer update retained in the final training trajectory. Calibration inputs, held-out evaluation, rolled-back timing probes, and auxiliary diagnostic forks do not contribute to this count.

The 1M budget contains exactly

$$1{,}048{,}576 \tag{39}$$

committed labels, corresponding to 64 full updates of $16{,}384$ labels. The exact 100K endpoint uses six full updates followed by a partial update of $1{,}696$ supervised labels.

Representation and ablation experiments use a frozen RedPajama CommonCrawl stream. The Qwen3-0.6B representation trajectory extends to 1M labels, while the Llama-3.2-1B representation comparison is evaluated through 100K labels. The representation results in Table 1 and Fig. 2 of the main paper use seed 0.

Scalar-QAT experiments use a fixed 1:1 mixture of SlimPajama and FineWeb-Edu. Vector-QAT experiments use a fixed English Ultra-FineWeb shard. Within each comparison track, methods consume the same supervised positions and ordering for a given seed.

Throughout the representation comparison, "active points" denotes the number of finite lattice/base points before applying the gain index. It is therefore a support-size statistic rather than the number of possible rank–gain records.

C.3 Optimization Settings

Table 7: Optimization settings used in the reported comparisons.

Method / track	LR	AdamW β	Weight decay	Schedule	GPUs
Representation BW / Product-E8 / JS-E8	1×10^{-5}	$(0.9, 0.95)$	0	cosine, 64 updates	1
Initialization ablation	1×10^{-5}	$(0.9, 0.95)$	0	cosine, 64 updates	1
Scalar BW-QAT	3×10^{-6}	$(0.9, 0.95)$	0	constant	1
StableQAT	3×10^{-4}	$(0.9, 0.999)$	0	constant	1
ParetoQ	3×10^{-5}	$(0.9, 0.999)$	0	constant	1
DSQ	3×10^{-5}	$(0.9, 0.999)$	0	constant	1
Vector BW-QAT	3×10^{-6}	$(0.9, 0.95)$	0	cosine, 64 updates	2
LC-QAT	3×10^{-5}	$(0.9, 0.95)$	source grouping	source WSD	2

All reported runs use AdamW with $\epsilon = 10^{-8}$, FP32 optimizer state, and global gradient-norm clipping at 1. For the scalar and vector comparisons, the learning rate is selected on a separate 100K-label development run and is then fixed for all reported seeds.

C.4 Evaluation Protocol

All held-out sets are frozen before model scoring. The representation comparison uses 128 spans with 2048 supervised labels per model. The initialization ablation uses a separate 64-span holdout. The scalar comparison uses 128 confirm spans of 2048 labels, while the vector comparison uses 64 confirm spans of 4096 labels. Forward computation uses BF16, and cross-entropy logits are evaluated in FP32.

WikiText-2 uses the `wikitext-2-raw-v1` test split. C4 uses fixed crops from its validation split. Perplexity is computed as the exponential of the label-weighted mean NLL.

Zero-shot downstream evaluation uses the LM Evaluation Harness with chat templates and automatic BOS disabled. Table 8 lists the metric used for each task and its aggregate membership.

Table 8: Downstream tasks used by Avg8-S and Avg6-LC. The "Avg6" column in Table 4 of the main paper refers to Avg6-LC.

Task	Metric	Avg8-S	Avg6-LC
ARC-Easy	acc_norm	✓	✓
ARC-Challenge	acc_norm	✓	✓
HellaSwag	acc_norm	✓	✓
PIQA	acc_norm	✓	✓
WinoGrande	acc	✓	✓
BoolQ	acc	✓	✓
OpenBookQA	acc_norm	✓	–
SciQ	acc_norm	✓	–

Avg8-S and Avg6-LC are unweighted arithmetic means over their constituent tasks. Aggregate task scores are reported in percent. Three-seed uncertainty is reported as the sample standard deviation with denominator $n - 1$.

C.5 Baseline Matching

The goal of each controlled comparison is to hold the data budget and evaluation conditions fixed while preserving the quantization mechanism of each method. Within a comparison track, methods therefore share the same base model, supervised stream, label budget, and evaluation protocol. The trainable state remains method-specific.

Table 9: Baseline configurations used in the controlled comparisons.

Method	Representation	Trainable state	Track	Preserved mechanism
Product-E8	finite Product-E_8	target proxies	representation	representation-specific codec / frozen gains
JS-E8	joint finite Product-E_8	target proxies	representation	joint finite codec / frozen gains
StableQAT	2-bit rowwise scalar	weights + scales	scalar	RDFS / source scale gradient
ParetoQ	2-bit rowwise scalar	weights + scales	scalar	StretchedElasticQuant
DSQ	2-bit rowwise scalar	weights + scales + α	scalar	source DSQ surrogate
LC-QAT	linear-constrained vector	source-defined state	vector	source parameterization / schedule

C.6 Native QAT Timing

The experimental server contains an Intel Core Ultra 9 285K CPU, 256 GB of system memory, and two NVIDIA RTX 5090 GPUs. Representation, initialization-ablation, and scalar-QAT runs use one GPU. The Qwen3-1.7B vector-QAT runs use two GPUs for both BW-QAT and LC-QAT.

Native QAT wall-clock time is measured over retained optimizer-update intervals. It includes forward and backward computation, gradient mapping and clipping, optimizer work, synchronization, and the required hard-state refresh. Model loading, initialization, held-out evaluation, deployment export, and independent reload are outside this timer.

Matched-time comparisons use the last complete optimizer-step endpoint within the fixed seed-0 BW-QAT wall-clock budget; no interpolation is used. The time reported in Table 4 of the main paper is the arithmetic mean over the three 1M-label runs, whereas Fig. 3 uses the corresponding seed-0 wall-clock measurements.

D Additional Results and Practical Cost

This section provides the per-seed values underlying the aggregate results in the main paper and gives the exact counting rule used by the initialization ablation.

D.1 Per-Seed Main Results

Table 10: Per-seed scalar-QAT results on Llama-3.2-1B at $1{,}048{,}576$ supervised labels.

Method	Seed	NLL ↓	Avg8-S ↑
BW-QAT	0	2.8897	51.90
	1	2.8930	51.86
	2	2.8907	51.95
StableQAT	0	8.0564	32.45
	1	7.9035	32.74
	2	7.7966	32.97
ParetoQ	0	7.4908	32.94
	1	7.2690	33.31
	2	7.4315	33.01
DSQ	0	8.1114	33.73
	1	8.1229	33.39
	2	8.3387	33.10

Table 11: Per-seed vector-QAT results on Qwen3-1.7B at $1{,}048{,}576$ supervised labels.

Method	Seed	NLL ↓	WT2 ↓	C4 ↓	Avg6-LC ↑
BW-QAT	0	2.6656	14.01	18.76	58.27
	1	2.6669	14.03	18.77	58.74
	2	2.6663	14.03	18.78	57.88
LC-QAT	0	2.9920	18.90	24.16	52.23
	1	2.9921	18.90	24.16	52.18
	2	2.9920	18.89	24.16	52.11

D.2 Initialization-Ablation Details

The ablation in Table 2 uses cumulative hard-record changes to measure whether the discrete model moves during QAT. A hard-record change is counted when the canonical $(\mathrm{rank}, \mathrm{gain})$ record of a target block changes between two consecutive committed optimizer updates. A simultaneous change in rank and gain counts as one record event. Counts are accumulated over all target blocks and all committed updates, including the transition from step zero to the first optimizer update.

The "fixed grid" terminology follows Table 2 in the main paper and denotes the fixed radius-based gain grid used without CCGC.

Table 12: Per-seed initialization ablation on Qwen3-0.6B at 100K supervised labels. Record events are cumulative across committed updates.

Variant	Seed	Q_0 NLL	100K NLL	Record events
CCGC + decision-preserving	0	3.9668	3.2914	358,788
	1	3.9668	3.3138	360,260
	2	3.9668	3.2785	359,937
CCGC + codeword init.	0	3.9668	3.9668	5
	1	3.9668	3.9668	5
	2	3.9668	3.9668	9
Fixed grid + decision-preserving	0	4.3351	3.2958	340,944
	1	4.3351	3.3251	335,783
	2	4.3351	3.2962	341,788

The two CCGC variants begin from the same hard model. Their different trajectories therefore isolate the effect of the continuous initialization. CCGC changes the quality of the step-zero hard model, while decision-preserving initialization changes the optimization state from which that hard model is updated.

D.3 Matched-Time Endpoints and Serialized Rates

Table 13 reports the exact seed-0 endpoints used by the matched-time analysis. Each comparison uses the same device allocation for all methods in that track.

Table 13: Seed-0 matched native-QAT-time endpoints. Each row is the last complete optimizer step within the BW-QAT wall-clock budget for that track.

Track	Method	GPUs	Wall time (s)	Labels	NLL ↓
Scalar	BW-QAT	1	5191.3	1.05M	2.8897
	StableQAT	1	5190.1	64.72M	3.9903
	ParetoQ	1	5189.5	35.27M	3.3604
	DSQ	1	5188.4	22.27M	5.7288
Vector	BW-QAT	2	4194.3	1.05M	2.6656
	LC-QAT	2	4193.1	14.73M	2.7466

The nominal target payload is 2 bits per weight. For BW-QAT, the realized target rates are 2.006 bpw on Llama-3.2-1B and 2.029 bpw on Qwen3-1.7B. The corresponding whole-model rates are 5.041 and 4.610 bpw. These values include the serialization overhead beyond the fixed 2-bit target records. The matched-time measurements above capture the training-throughput trade-off reported in the main paper.