MPQ: A Message-Passing View of Post-Training Quantization

Yanshuo Chen, Chenxi Liu, Reza Shirkavand, Heng Huang
Department of Computer Science, University of Maryland, College Park
{cys,cxliu539,rezashkv,heng}@umd.edu

Abstract

Post-training quantization (PTQ) is a key technique for efficiently deploying large language models, as it compresses weights into low-bit representations without retraining. Existing PTQ solvers, including GPTQ, LDLQ, and block coordinate descent, commit each weight block to a single codeword before its neighbors are considered, leaving near-optimal alternatives unexplored. In this work, we cast row-wise PTQ as posterior inference over discrete codebooks, in which existing solvers correspond to a zero-temperature, hard-decision limit retaining only the top codeword per block. We propose *Message-Passing Quantization* (MPQ), which preserves soft posterior information across blocks during inference and commits to hard codewords only at the output stage. MPQ first uses vector approximate message passing to compute approximate blockwise posteriors, then refines a hard-decision initialization by exact block coordinate descent. Across 2-bit lattice and 3- and 4-bit scalar quantization on dense language models from Llama, Qwen, and Mistral families, MPQ improves perplexity and zero-shot accuracy over strong hard-decision baselines, and composes favorably with preprocessing techniques such as activation-aware scaling and incoherence processing. Our implementation is available at https://github.com/poseidonchan/MPQ.

1 Introduction

The growing capabilities of large language models have made them increasingly useful, motivating efficient deployment on personal devices [1, 2, 3, 4, 5]. Post-training quantization (PTQ) compresses a pretrained language model by mapping full-precision weights to low-bit discrete codes, enabling deployment under tight memory and compute budgets without additional retraining. Recent progress has mainly improved the preprocessing and objective-design stage of PTQ: activation scaling methods reduce quantization difficulty by rescaling weights and activations [6, 7, 8, 9]; incoherence processing methods make weights easier to quantize through rotations or related transformations [10, 11]; structured codebooks improve the expressiveness of low-bit representations [12, 11]; and inter-layer error propagation objectives better account for how quantization errors affect later layers [13, 14].

In contrast, the discrete solver stage has changed more incrementally. Most existing solvers quantize blocks sequentially, make an immediate hard assignment to a single codeword, and then greedily compensate for the resulting error. GPTQ, LDLQ, and later block coordinate descent variants differ in scheduling and local approximation, but they remain fundamentally hard-decision solvers [15, 10, 16, 17, 18]. This approach is efficient but commits to a single hard codeword per block before its neighbors are processed.

This early commitment is especially harmful in the low-bit regime. With very few available codes, several codewords can produce nearly the same local error, and strong Hessian coupling makes the best choice for one block depend on its neighbors. A hard-decision solver therefore commits to one codeword trajectory through the discrete assignment space, with no mechanism to recover if

40th Conference on Neural Information Processing Systems (NeurIPS 2026).

those early choices turn out to be globally suboptimal. As preprocessing, codebook, and objective design improve the conditioning of the quantization problem, the discrete solver remains a relatively underexplored axis. Instead of assigning each block to a single codeword immediately, the solver should maintain a soft distribution over plausible codewords for each block and let these distributions interact with their neighbors before any block commits to a hard decision.

We take a message-passing view of row-wise PTQ, casting it as posterior inference over a discrete codebook with a Gaussian likelihood. From this view, existing solvers such as GPTQ, LDLQ, and block coordinate descent are zero-temperature message-passing schemes that propagate point estimates between blocks and discard other plausible codewords. This perspective naturally suggests a finite-temperature alternative, where messages carry distributions over several likely codewords before any block commits.

We instantiate this idea with *Message-Passing Quantization* (MPQ), a two-stage solver that first explores uncertainty and then commits to a discrete solution. In the first stage, MPQ uses vector approximate message passing (VAMP) at finite temperature to aggregate information about near-tied codeword assignments across blocks. In the second stage, it converts the resulting posterior into a hard initialization and refines it with block coordinate descent. Experiments on Llama, Qwen, and Mistral models show that MPQ matches or improves over greedy solver baselines, with the largest gains at 3-bit scalar and 2-bit lattice precision.

Our contributions are as follows.

- We give a message-passing view of row-wise PTQ as posterior inference over a discrete codebook under a Gaussian likelihood, recovering existing solvers as zero-temperature limits and motivating finite-temperature inference as an unexplored solver-side direction.
- We propose Message-Passing Quantization (MPQ), a two-stage solver that combines finite-temperature vector approximate message passing with exact block coordinate descent.
- MPQ improves over strong solver baselines in perplexity and zero-shot accuracy across Llama, Qwen, and Mistral at 2-, 3-, and 4-bit quantization, with the largest gains in the lowest-bit regimes
- We complement the method with theoretical analysis of the VAMP solver's asymptotic behavior and the role of preprocessing in finite-temperature inference.

2 Related Work

Preprocessing, codebook, and objective design. Many studies improve the quantization problem before the solver runs. Scaling and equalization methods such as AWQ, SmoothQuant, OmniQuant, and MagR reduce dynamic-range imbalance or protect salient channels before rounding [6, 7, 19, 20]. Rotation and incoherence methods such as QuIP, QuIP#, QuaRot, SpinQuant, and QTIP reshape the Hessian geometry using random or learned orthogonal transforms [10, 11, 8, 9, 12]. Fine-tuning methods such as PV-Tuning further improve the compressed representation after quantization [21]. YAQA and Qronos extend layer-wise PTQ to account for errors propagating across layers [13, 14].

Solver-side PTQ. GPTQ made layerwise second-order quantization practical at LLM scale by developing OBQ-style formulations into an efficient column-wise pass [15]. QuIP and QuIP# extended this to vector quantization with LDLQ and incoherence processing [10, 11]. QuantEase and CDQuant revisit local repair via explicit block coordinate descent [16, 17]. Recent theoretical work has shown that GPTQ is equivalent to Babai's nearest-plane algorithm on the quantization lattice [22, 23]. Building on this lattice perspective, OJBKQ further augments greedy Babai rounding with Klein's sampling to sample multiple stochastic trajectories in the search space [18]. While Klein sampling injects stochasticity across multiple Babai trajectories, each individual trajectory still commits to one codeword per block sequentially. Section 3 unifies these solvers as zero-temperature limits of a common posterior view.

Message passing for discrete inference. Row wise PTQ is a linear inverse problem with discrete constraints. Methods in the approximate message passing family maintain soft beliefs over latent blocks rather than collapsing immediately to point estimates [24, 25, 26]. VAMP extends AMP with rigorous state evolution for large right orthogonally invariant operators [27]. Although the random Hadamard transform used in modern PTQ pipelines does not make a finite calibration Hessian exactly right orthogonally invariant, it randomizes the coordinate basis and improves incoherence, making

the right orthogonally invariant model a useful surrogate for analysis [10, 11]. We therefore use VAMP as the finite temperature inference stage of MPQ and interpret the associated state evolution as qualitative guidance for the RHT conditioned PTQ problem.

3 Preliminaries

PTQ formulation. Let $W \in \mathbb{R}^{m_{\text{out}} \times n}$ be the weight matrix of a single linear layer and $X \in \mathbb{R}^{m \times n}$ a matrix of calibration activations. Weight-only post-training quantization replaces W with a quantized matrix $\hat{W}$ by minimizing the layer-wise reconstruction objective $\|X(W - \hat{W})^\top\|_F^2$, which decomposes over the m_{out} output rows.

We identify each output row of W with its transpose, so a single row is written as a column vector $w \in \mathbb{R}^n$. We partition the n coordinates into $B = n/d$ non-overlapping blocks of dimension d, assuming d divides n (or after grouping/padding otherwise). Each block b is constrained to a finite codebook $\mathcal{C}_b \subset \mathbb{R}^d$ of size M_b; this covers both uniform scalar grids ($d = 1$) and structured vector codebooks such as E8P12 ($d = 8$, $M_b = 2^{16}$) [11]. Writing $\hat{w} = (\hat{w}_1, \ldots, \hat{w}_B)$ with $\hat{w}_b \in \mathcal{C}_b$, the per-row problem is

$$\min_{\hat{w}_1 \in \mathcal{C}_1,\, \ldots,\, \hat{w}_B \in \mathcal{C}_B} \ \frac{1}{2}\big\|Xw - X\hat{w}\big\|_2^2. \tag{1}$$

Because the quadratic form $\frac{1}{2}\hat{w}^\top(X^\top X)\hat{w}$ couples all blocks through the off-diagonal structure of $X^\top X$, changing one block perturbs the residual seen by every other, making it a combinatorial problem over $\prod_b M_b$ configurations.

A unified probabilistic view. Fix an output row $w = (w_1, \ldots, w_B)$, and let $E_w(\hat{w}) := \frac{1}{2}\|Xw - X\hat{w}\|_2^2$ denote its per-row contribution to the layer-wise reconstruction objective on the product codebook $\mathcal{C}_{\text{prod}} = \mathcal{C}_1 \times \cdots \times \mathcal{C}_B$. We introduce the finite-temperature Gibbs measure

$$\begin{aligned} p_\beta(\hat{w} \mid w, X) &= \frac{1}{Z_\beta} \exp\{-\beta E_w(\hat{w})\} \prod_{b=1}^{B} \rho_b(\hat{w}_b), \\ \rho_b(c) &\propto \exp\left\{-\frac{\|c - w_b\|_2^2}{2\tau^2\sigma_b^2}\right\}, \qquad c \in \mathcal{C}_b. \end{aligned} \tag{2}$$

Equivalently, this Gibbs measure is the posterior induced by the Gaussian likelihood $Xw \mid \hat{w} \sim \mathcal{N}(X\hat{w}, \beta^{-1} I_m)$ and the codebook-supported reference measure $\prod_{b=1}^{B} \rho_b$, where β is the inverse temperature. The reference measure has anchoring width $\tau\sigma_b$ formed from a fixed per-block scale σ_b and a global multiplier τ that broadens the prior uniformly across blocks.

Since $\mathcal{C}_{\text{prod}}$ is finite and $\rho_b(c) > 0$ for all $c \in \mathcal{C}_b$, the zero-temperature limit $\beta \to \infty$ concentrates on the minimizer set of Eq. (1). Thus, in the zero-temperature limit, MAP inference under this Gibbs measure recovers the minimizers of the original per-row PTQ problem in Eq. (1). Hard-assignment PTQ solvers such as GPTQ, LDLQ, and block coordinate descent can be viewed as approximations to this zero-temperature MAP problem, differing in how they schedule blocks and how they account for inter-block coupling. At finite β, the posterior retains blockwise means, variances, and probability mass on near-best codewords, which our message-passing solver propagates.

4 Message-Passing Quantization

MPQ instantiates finite-temperature inference on the Gibbs posterior in Eq. (2) and then refines the resulting hard assignment by exact block coordinate descent on the row-wise PTQ objective in Eq. (1). The first stage preserves uncertainty over near-tied codewords before commitment, and the second stage performs exact local repair after projection to a discrete initialization.

4.1 Finite-temperature VAMP in rank space

Let $A := X$ and $y := Xw$ with $x \in \mathbb{R}^n$ denoting the quantized row $\hat{w}$ in VAMP notation, so that the row-wise problem (Eq. (1)) becomes $\min_{x \in \mathcal{C}_{\text{prod}}} \|y - Ax\|_2^2$. Because the objective depends on

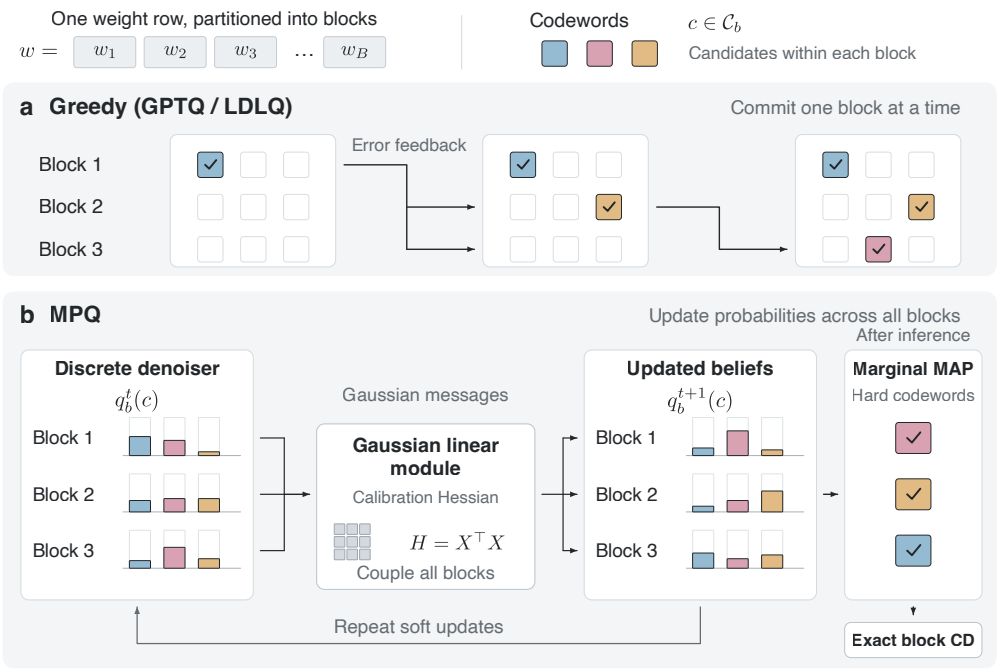

Figure 1: Greedy quantization and MPQ. Rows represent weight blocks b. Colors identify codewords $c \in \mathcal{C}_b$, and bar heights show probabilities $q_b^t(c)$ (Eq. (4)). (a) Greedy solvers fix blocks sequentially and pass error to remaining blocks. (b) MPQ alternates denoising and Gaussian linear updates across all blocks, then applies marginal MAP projection and exact block CD.

x only through the rank-r coordinates $\Sigma_A V^\top x$, the first stage can be implemented entirely in rank space. With the compact SVD $A = U\Sigma_A V^\top$, where $\Sigma_A = \mathrm{diag}(s_1, \ldots, s_r)$ and $r = \mathrm{rank}(A)$, set $T := \Sigma_A V^\top$ and $\tilde{y} := U^\top y$, so that

$$\|y - Ax\|_2^2 = \|\tilde{y} - Tx\|_2^2 + \mathrm{const}, \tag{3}$$

and all first-stage computations use $T \in \mathbb{R}^{r \times n}$ rather than the original calibration matrix.

At iteration t, VAMP alternates between a blockwise discrete denoiser and a Gaussian linear module. Given a Gaussian cavity (r_1^t, γ_1^t), with $r_{1,b}^t \in \mathbb{R}^d$ denoting the b-th block of r_1^t, the denoiser updates each block independently via

$$q_b^t(c) \propto \rho_b(c) \exp\left(-\frac{\gamma_1^t}{2}\|c - r_{1,b}^t\|_2^2\right), \qquad c \in \mathcal{C}_b, \tag{4}$$

with blockwise posterior mean and covariance

$$\hat{x}_{1,b}^t = \sum_{c \in \mathcal{C}_b} c\, q_b^t(c), \qquad \Sigma_{1,b}^t = \sum_{c \in \mathcal{C}_b} (c - \hat{x}_{1,b}^t)(c - \hat{x}_{1,b}^t)^\top q_b^t(c), \tag{5}$$

and average sensitivity

$$\alpha_1^t = \frac{\gamma_1^t}{n} \sum_{b=1}^{B} \mathrm{tr}\left(\Sigma_{1,b}^t\right). \tag{6}$$

The standard VAMP extrinsic update then converts $(\hat{x}_1^t, \alpha_1^t)$ into the Gaussian message (r_2^t, γ_2^t) for the linear module; exact formulas are deferred to Appendix B.1.

With $H := A^\top A$ and $g := A^\top y$, the linear module solves the Gaussian quadratic subproblem $\hat{x}_2^t = (\beta^t H + \gamma_2^t I)^{-1}(\beta^t g + \gamma_2^t r_2^t)$ at the current empirical-Bayes inverse temperature β^t (adapted within iterations, see Appendix B.1), which in rank space evaluates to

$$\hat{x}_2^t = r_2^t + V\, \mathrm{diag}\left(\left\{\frac{\beta^t s_\ell}{\beta^t s_\ell^2 + \gamma_2^t}\right\}_{\ell=1}^{r}\right)\left(\tilde{y} - \Sigma_A V^\top r_2^t\right), \tag{7}$$

Algorithm 1 MPQ for one output row

Require: rank-space $(T, \tilde{y})$, row w, codebooks $\{\mathcal{C}_b\}$, prior scales $\{\sigma_b\}$, anchoring multiplier τ, VAMP iterations T_{vamp}, CD budget S_{CD}

Initialize $(r_1^0, \gamma_1^0, \beta^0)$ as in Appendix B.1
for $t = 0, \ldots, T_{\mathrm{vamp}} - 1$ **do**
  Denoiser: for each block b, form q_b^t (4) and compute $(\hat{x}_{1,b}^t, \Sigma_{1,b}^t)$ (5); aggregate α_1^t (6)
  Extrinsic: apply (19) to form (r_2^t, γ_2^t)
  Linear: compute $\hat{x}_2^t$ (7) and α_2^t (8); update β^{t+1} (22)
  Extrinsic: apply (20) to form $(r_1^{t+1}, \gamma_1^{t+1})$
end for
MAP projection: for each block b, set $x_b \leftarrow \arg\max_{c \in \mathcal{C}_b} q_b^{T_{\mathrm{vamp}}}(c)$
Set $u \leftarrow \tilde{y} - Tx$
for $p = 1, \ldots, S_{\mathrm{CD}}$ **do**
  For each b, set $x_b^{\mathrm{old}} \leftarrow x_b$, update x_b by (12), and update $u \leftarrow u - T_b(x_b - x_b^{\mathrm{old}})$
  break if no block changed during this sweep
end for
return $(x_1, \ldots, x_B)$

with average sensitivity

$$\alpha_2^t = \frac{1}{n} \operatorname{tr}\left[\gamma_2^t (\beta^t H + \gamma_2^t I)^{-1}\right]. \tag{8}$$

A second extrinsic update then maps $(\hat{x}_2^t, \alpha_2^t)$ back to $(r_1^{t+1}, \gamma_1^{t+1})$, again with exact formulas in Appendix B.1. In the SVD form, the null-space component of r_2^t is left unchanged, corresponding to the $(n-r)/n$ contribution implicit in Eq. (8) when $r < n$. We repeat these denoiser and linear-module updates for T_{vamp} iterations, keeping every block soft-valued throughout the first stage.

4.2 Zero-temperature exact local refinement

After T_{vamp} iterations, MPQ forms one final denoiser posterior at $(r_1^{T_{\mathrm{vamp}}}, \gamma_1^{T_{\mathrm{vamp}}})$ and projects it to a hard initialization by blockwise marginal MAP,

$$x_b^{(0)} = \arg\max_{c \in \mathcal{C}_b} q_b^{T_{\mathrm{vamp}}}(c). \tag{9}$$

MPQ uses blockwise marginal modes, which need not form a joint MAP assignment because the Hessian couples the blocks. It then runs exact block coordinate descent (CD) on the original PTQ objective in Eq. (1), for up to S_{CD} sweeps with early stopping. Let

$$u := \tilde{y} - Tx \tag{10}$$

be the current residual, and let T_b denote the columns of T associated with block b. When block b is set to a candidate $c \in \mathcal{C}_b$, the resulting objective value is

$$F_b(c) = \frac{1}{2} \|u - T_b(c - x_b)\|_2^2. \tag{11}$$

The block update is therefore

$$x_b \leftarrow \arg\min_{c \in \mathcal{C}_b} F_b(c). \tag{12}$$

This search is exact within each block, but it is still local because each update is evaluated against the current residual. Appendix B.4 formalizes this as nearest-codeword search in a Mahalanobis metric whose center moves with the iteration. The VAMP warm start is therefore useful not because it replaces local search, but because it enters refinement from a better basin.

Algorithm 1 summarizes the row-wise MPQ procedure at the level of the mathematical updates. A Python-style pseudocode implementation is given in Appendix B.5.

5 Theoretical Analysis

The key insight of MPQ is that VAMP keeps near-tie information that hard solvers discard. We analyze this effect in an idealized ROI surrogate at the unanchored $\tau \to \infty$ limit, tracking the

denoiser-side fixed-point precision $\gamma_1^\star$. Theorem 5.1 bounds the MAP projection error probability by an exponential in $\gamma_1^\star$, and Proposition 5.2 derives an exact formula for how diagonal preprocessing changes the linear-module error E_{lin}, which through state evolution affects $\gamma_1^\star$. The two results together trace a chain from preprocessing to MAP accuracy. MPQ itself runs at finite τ, so we treat this analysis as qualitative guidance rather than a tight characterization.

5.1 Posterior concentration under VAMP

Consider the blockwise surrogate model

$$y = Ax^\star + \xi, \qquad \xi \sim \mathcal{N}(0, \beta_0^{-1} I), \tag{13}$$

where y is the noisy observation, $x^\star = [C_1^\star; \ldots; C_B^\star]$, $C_b^\star \overset{\text{i.i.d.}}{\sim} \mathrm{Unif}(\mathcal{C})$ for a finite codebook $\mathcal{C} \subset \mathbb{R}^d$, and A is ROI. Under VAMP state evolution, let $(\gamma_1^\star, \gamma_2^\star)$ be a locally stable fixed point reached by damped VAMP [27, Theorem 1]; Appendix E.1 specializes the recursion to our blockwise discrete denoiser. At this fixed point, each block sees an effective additive white Gaussian noise (AWGN) channel of precision $\gamma_1^\star$, and the following theorem bounds the per-block MAP error from that channel.

Theorem 5.1 (Posterior concentration and warm-start accuracy)**.** *At the state-evolution fixed point, for any block, the effective channel can be written as*

$$R = C^\star + \eta, \qquad \eta \sim \mathcal{N}\big(0, (\gamma_1^\star)^{-1} I_d\big), \tag{14}$$

where $C^\star \sim \mathrm{Unif}(\mathcal{C})$ is the true codeword, indexed as $C^\star = c^{(j^\star)}$. For each j, define the score

$$s_j(R) := -\frac{\gamma_1^\star}{2} \|R - c^{(j)}\|_2^2,$$

and define $\delta_j := c^{(j)} - c^{(j^\star)}$ for each competitor $j \neq j^\star$. Then the blockwise MAP error $\hat{j}_{\mathrm{MAP}} := \arg\max_j s_j(R)$ obeys

$$\mathbb{P}\big(\hat{j}_{\mathrm{MAP}} \neq j^\star\big) \leq \sum_{j \neq j^\star} \exp\left(-\frac{\gamma_1^\star \|\delta_j\|_2^2}{8}\right). \tag{15}$$

Proof sketch. For each competitor j, the score difference $s_j(R) - s_{j^\star}(R)$ is Gaussian with mean $-\frac{\gamma_1^\star}{2}\|\delta_j\|_2^2 < 0$ and variance $\gamma_1^\star \|\delta_j\|_2^2$, so the standard Gaussian tail bound controls the probability that this competitor outranks the true codeword. A union bound over competitors yields Eq. (15). Appendix E.2 contains the full argument.

□

Remark. The bound (15) makes the intuition developed in Section 1 more precise. When codewords are close in Euclidean distance, the pairwise gaps Δ_j are small and the bound is loose. As $\gamma_1^\star$ increases, whether through better conditioning or preprocessing, the exponential tails shrink and the bound certifies a reliable MAP projection. Conversely, when the codebook is dense and many nearby competitors remain, the bound is no longer informative, and any residual error in this regime is left to the block coordinate descent. Section 6.5 reports diagnostics consistent with this prediction, showing that the dense E8P12 codebook yields a smaller terminal precision proxy and a substantially larger fraction of blocks updated by CD than the scalar codebook.

5.2 Preprocessing and the linear module

Theorem 5.1 shows that a higher effective precision $\gamma_1^\star$ yields a sharper warm start. Through state evolution (Appendix E.1), $\gamma_1^\star$ is determined jointly by the denoiser MSE on the uniform codebook prior and the linear-module error

$$E_{\mathrm{lin}}(H, \gamma) := \frac{1}{n} \operatorname{tr}\big[(\beta_0 H + \gamma I)^{-1}\big], \tag{16}$$

which depends on the Hessian spectrum (β_0 is introduced in Eq. (13)). In this surrogate, fixing the codebook geometry, preprocessing enters the message-passing linear module entirely through E_{lin}. We characterize this dependence for diagonal preprocessing such as AWQ [6].

Let $H_0 \succeq 0$ be a fixed baseline Hessian and let $D = \mathrm{diag}(a_1, \ldots, a_n) \succ 0$ be a positive diagonal scaling. We interpolate from H_0 to DH_0D along the path $H_t := D_t H_0 D_t$ with $D_t := \exp(tL)$ and $L := \log D$, so that $D_0 = I$ and $D_1 = D$.

Proposition 5.2 (Preprocessing effect on the linear module)**.** *For every $\gamma > 0$, the path $t \mapsto E_{\mathrm{lin}}(H_t, \gamma)$ is continuously differentiable and satisfies*

$$\frac{d}{dt} E_{\mathrm{lin}}(H_t, \gamma) = -\frac{2\beta_0}{n} \operatorname{tr}\left[L\, H_t \left(\beta_0 H_t + \gamma I\right)^{-2}\right]. \tag{17}$$

Consequently,

$$E_{\mathrm{lin}}(H_1, \gamma) - E_{\mathrm{lin}}(H_0, \gamma) = -\frac{2\beta_0}{n} \int_0^1 \operatorname{tr}\left[L\, H_t \left(\beta_0 H_t + \gamma I\right)^{-2}\right] dt. \tag{18}$$

Proof sketch. Differentiate $H_t = D_t H_0 D_t$ to obtain $\dot{H}_t = LH_t + H_t L$. Then differentiate the resolvent $(\beta_0 H_t + \gamma I)^{-1}$ with the inverse-derivative identity and use that H_t commutes with any polynomial in itself, in particular with $(\beta_0 H_t + \gamma I)^{-1}$. Cyclicity of trace yields Eq. (17), and integrating from $t = 0$ to $t = 1$ gives Eq. (18). Appendix E.3 contains the full derivation. □

Remark. The integral (18) captures the effect of AWQ in isolation, but the production pipeline composes AWQ with a random Hadamard transformation (RHT) O, replacing H_t by the conjugated Hessian $O^\top H_t O$. Since E_{lin} depends on the Hessian only through its eigenvalues, this conjugation leaves E_{lin} unchanged (Lemma E.2), and the analysis of ΔE_{lin} along the AWQ path applies verbatim to the AWQ+RHT pipeline. For clear effect isolation, Corollary E.3 rewrites the integral as a sum over coordinates of nonnegative resolvent-weighted Hessian factors κ_i scaled by the log-scales ℓ_i, making the sign explicit. Coordinates with $\ell_i > 0$ decrease E_{lin} while those with $\ell_i < 0$ increase it, with the magnitude of each contribution weighted by κ_i. The diagnostic in Section 6.5 evaluates this signed quantity on the realized AWQ scales and shows that the net contribution is typically negative, consistent with AWQ lowering E_{lin} in the preprocessing pipeline.

6 Experiments

6.1 Setup

Protocol. We use 128 C4 [28] training-set sequences of length 2048 as calibration data to quantize dense language model weights, while keeping activations at 16 bits during evaluation. We measure WikiText-2 [29] and C4 perplexity and zero-shot normalized accuracy on ARC-Challenge, ARC-Easy [30], HellaSwag [31], PIQA [32], and Winogrande [33] using `lm-eval` harness [34]. We also repeat the 3-bit and 4-bit scalar experiments on four Llama models with three independently sampled C4 calibration sets. Methods share each calibration draw. Appendix A.7 reports mean perplexity and sample standard deviation.

Codebook and solvers. For 3/4-bit scalar codebook quantization, we use a group size of 128 and apply the activation-aware scaling [6] followed by RHT [10] as preprocessing for all solvers. For the GPTQ baseline, we use the strongest variant from GPTQ [15] with activation-ordered column processing. For 2-bit quantization, we adopt vector quantization with the E8P12 codebook [11], which provides better rate-distortion than scalar codebooks at this aggressive bit width. Following the standard recipe in QuIP# [11], we employ RHT for incoherence preprocessing, and take LDLQ as the corresponding greedy baseline [11]. For each greedy solver, GPTQ in the scalar setting and LDLQ in the lattice setting, we also include a stronger baseline that applies 3 passes of block coordinate descent (CD), matching the CD refinement used by MPQ. Additionally, we include QuantEase [16] and OJBKQ [18] as strong CD and randomized greedy baselines, respectively. Our MPQ solver's default setting is 16 VAMP iterations followed by 3 CD passes. Appendix A.1 gives the detailed hyperparameter settings for MPQ.

6.2 Llama-family perplexity scaling

We first compare MPQ with baselines inside the Llama family across different model sizes [35, 36]. Table 1 shows that MPQ improves over the strongest baseline OJBKQ with the magnitude of the

Table 1: Llama-family PTQ perplexity scaling.

Method	Llama3.2-1B		Llama3.2-3B		Llama3-8B		Llama2-13B		Llama2-70B	
	Wiki	C4	Wiki	C4	Wiki	C4	Wiki	C4	Wiki	C4
BF16	9.75	12.77	7.82	10.47	6.14	8.93	4.89	6.47	3.32	5.52
4-bit scalar codebook, group size 128										
GPTQ	10.63	13.86	8.22	11.08	6.54	9.41	4.99	6.57	3.43	5.58
GPTQ + CD	10.57	13.83	8.24	11.10	6.55	9.42	4.98	6.57	3.43	5.58
QuantEase	10.47	13.80	8.18	11.03	6.50	9.37	4.98	6.57	3.42	5.58
OJBKQ	10.43	13.68	8.15	10.99	6.48	9.35	4.97	6.56	3.42	5.58
MPQ	**10.38**	**13.63**	**8.12**	**10.94**	**6.46**	**9.33**	**4.96**	**6.55**	**3.41**	**5.57**
3-bit scalar codebook, group size 128										
GPTQ	15.45	19.25	10.56	13.76	8.29	11.61	5.46	7.09	3.93	5.89
GPTQ + CD	15.58	18.94	10.51	13.82	8.43	11.63	5.46	7.09	3.92	5.90
QuantEase	14.86	18.51	10.05	13.36	8.10	11.24	5.39	6.95	3.88	5.88
OJBKQ	14.26	18.00	9.93	13.22	7.95	11.08	5.40	6.98	3.87	5.86
MPQ	**13.65**	**17.42**	**9.65**	**12.87**	**7.81**	**10.88**	**5.30**	**6.89**	**3.78**	**5.80**
2-bit E8P12 codebook										
LDLQ	42.80	43.47	18.54	21.15	15.01	18.17	6.63	8.17	4.58	6.36
LDLQ + CD	41.79	42.69	18.00	20.66	12.76	15.96	6.61	8.13	4.59	6.36
QuantEase	47.31	46.24	18.84	21.17	14.78	17.77	6.67	8.19	4.70	6.46
OJBKQ	39.28	39.72	17.55	19.94	13.84	16.57	6.55	8.11	4.58	6.36
MPQ	**31.67**	**31.48**	**15.38**	**18.10**	**11.63**	**14.77**	**6.27**	**7.76**	**4.49**	**6.30**

Table 2: Average zero-shot accuracy in percentage points. Bold denotes the best quantized result within each model and bit width. Underlining denotes second best. Table 4 gives the detailed accuracy for each task.

Model	BF16	3-bit scalar					2-bit E8P12				
		GPTQ	GPTQ+CD	QuantEase	OJBKQ	MPQ	LDLQ	LDLQ+CD	QuantEase	OJBKQ	MPQ
Mistral-7B	74.3	71.2	71.2	**72.1**	<u>71.9</u>	71.9	66.8	<u>67.5</u>	66.1	<u>67.5</u>	**67.9**
Qwen3-8B	73.5	69.3	70.1	70.4	<u>71.0</u>	**71.8**	<u>68.1</u>	67.2	63.5	66.5	**68.4**
Llama3-8B	73.0	67.9	67.7	68.1	<u>68.5</u>	**68.6**	57.4	<u>59.3</u>	55.3	59.2	**59.7**
Llama2-13B	71.6	69.1	69.4	69.9	<u>70.0</u>	**70.1**	65.5	65.5	65.8	<u>66.0</u>	**66.9**
Llama2-70B	76.7	**75.3**	74.9	75.0	<u>75.2</u>	**75.3**	73.7	73.5	<u>74.2</u>	<u>74.2</u>	**74.6**

gain governed by the bit budget. At 4 bits, the baselines are already close to the BF16 ceiling and MPQ's lead is small but consistent. The gap widens at 3-bit scalar and widens further at 2-bit E8P12, where the greedy baselines degrade sharply. The GPTQ plus CD and LDLQ plus CD rows isolate the effect of the polishing stage. MPQ remains ahead when all initializations receive the same three CD passes. In the 2-bit lattice regime, its hard projection can start with a higher objective than the greedy initialization and reach a lower objective after the same CD refinement. The absolute gap narrows with model scale, consistent with larger models being more robust to quantization noise, but the ranking is preserved across all model sizes and three precision budgets.

6.3 Zero-shot transfer across model families

We next check whether MPQ's perplexity gains carry over to downstream accuracy. Table 2 reports the five task average on Mistral 7B [37], Qwen3 8B [38], Llama 3 8B [36], Llama 2 13B, and Llama 2 70B [35], covering both the 3-bit scalar and 2-bit E8P12 regimes. MPQ improves over the matched CD polished greedy baseline in every model and bit width, with gains ranging from 0.4 to 1.7 percentage points. Against the stronger QuantEase and OJBKQ baselines, MPQ is best or tied best in nine of ten model/bit width cells. The only exception is Mistral 7B at 3-bit scalar, where QuantEase leads by 0.2 points and MPQ ties OJBKQ for second best. At 2-bit E8P12, MPQ is the best method for all five models, improving over LDLQ+CD by 0.4 to 1.4 points. These results show that MPQ's advantage is not restricted to perplexity and remains visible on downstream multiple choice evaluation, especially in the more constrained 2-bit lattice regime.

6.4 Solver scalability

Timing and memory protocol. Following the experimental setup in Section 6.1, we measure all solver-side timings on a single RTX 5090, excluding preprocessing, model export, and evaluation, so that the reported numbers reflect the cost of the solver itself. We use `torch.compile` and set the number of parallel blocks to 1024 throughout to speed up the time-consuming block CD. Peak memory is reported as the CUDA allocated peak during quantization.

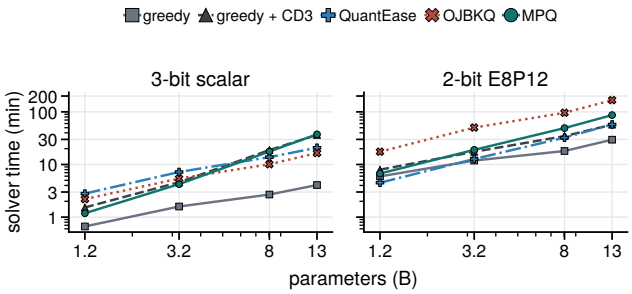

Figure 2: Solver-time scaling across Llama model sizes.

Solver-time scaling across model sizes. Figure 2 traces solver-time scaling across four Llama-family scales from 1B to 13B parameters, with detailed numbers in Table 6. In the 3-bit scalar regime, MPQ matches GPTQ+CD3 across all four scales, but remains several times slower than the greedy GPTQ baseline. In the 2-bit E8P12 regime, this gap narrows substantially. MPQ stays within 3× of LDLQ and is faster than the strongest baseline OJBKQ across all scales. These time and memory costs arise only during offline PTQ.

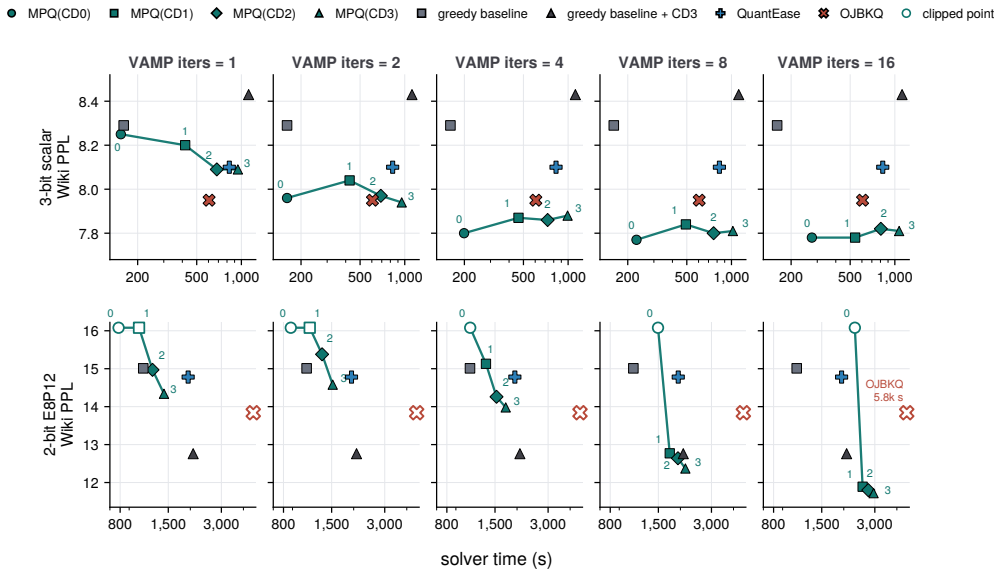

Figure 3: Solver-side speed-quality tradeoff on Meta-Llama-3-8B.

Speed-quality tradeoff on Llama-3-8B Figure 3 fixes Meta-Llama-3-8B and shows how PPL changes with solver time as VAMP iterations vary across columns and CD passes across markers. Detailed numbers and memory usage are reported in Appendix A.5. In the 3-bit scalar regime, 4 VAMP iterations with no CD already reach 7.8 WikiText-2 PPL, below both OJBKQ and QuantEase, and extra CD sweeps barely move the curve. Users in this regime can drop CD entirely and recover most of the wall-clock budget reported in Figure 2, since VAMP alone already saturates scalar quality. In the 2-bit E8P12 regime, VAMP-only configurations clip above 16 PPL, but each additional CD sweep pushes the operating point lower, reaching around 11.6 PPL at 16 VAMP iterations with three CD passes against LDLQ+CD3 at 12.76. The CD budget is therefore a quality dial that users can spend when targeting aggressive low-bit deployments.

Matched solver budgets. On Llama-3-8B, MPQ with eight VAMP iterations and three CD sweeps reaches 12.370 WikiText-2 PPL in 2134 seconds, compared with 12.756 in 2076 seconds for

LDLQ+CD3. With sixteen VAMP iterations and three CD sweeps, MPQ reaches 11.630 in 2951 seconds, compared with 12.728 in 3057 seconds for LDLQ+CD6 (Table 13).

Deployment. MPQ quantizes weights offline into GPTQ-compatible scalar or QuIP# E8P12 formats. With matched grouping and packing, the quantized models use the corresponding inference kernels.

6.5 Ablation and mechanism study

Appendices A.1 and A.4 cover hyperparameter sensitivity and activation scaling. We next examine codebook choice, CD budget, and the precision and preprocessing diagnostics motivated by Section 5.

Codebook and refinement controls. With the 2-bit D4 codebook, MPQ has the lowest objective after one CD sweep and the lowest perplexity after three sweeps (Appendix A.8). With E8P12, MPQ has lower perplexity than LDLQ after 20 CD sweeps (Table 14).

VAMP precision and CD cleanup. Figure 4a connects the empirical gains to the warm-start picture in Theorem 5.1 by plotting the final VAMP precision proxy $\hat{\gamma}_1$ against the fraction of assignments changed by CD after VAMP from each module. The 3-bit scalar regime sits at high precision with negligible cleanup, while the 2-bit scalar regime has much higher cleanup ratio. Moreover, the 2-bit E8P12 regime has much lower precision and higher cleanup ratio than 2-bit scalar regime. This pattern indicates that the message-passing stage is near terminal in 3-bit scalar regimes but mainly selects a basin in the harder 2-bit scalar and lattice codebook regime, leaving the precise codeword commitment to CD. Appendix A.6 gives the per-model and greedy-baseline cleanup audits.

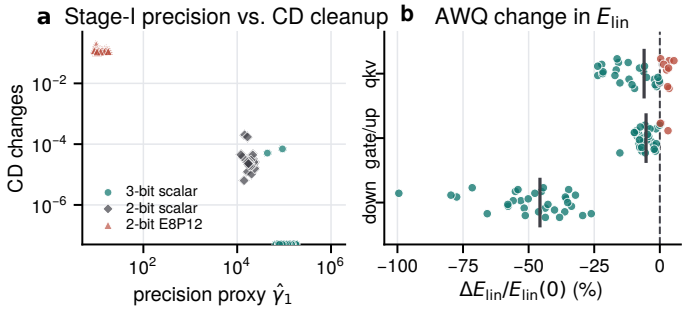

Figure 4: Mechanism diagnostics on Meta-Llama-3-8B.

AWQ and the linear proxy. Figure 4b tests whether AWQ moves the linear proxy in the direction predicted by Proposition 5.2 under the same AWQ+RHT scalar setup on Meta-Llama-3-8B. AWQ decreases E_{lin} for most scale groups, with the few increases concentrated in `qkv` and `gate_up`, and the sign of the leading-order integrand from Proposition 5.2 agrees with the observed change on nearly all groups. Appendix A.6 gives the full sign-count breakdown.

7 Conclusion

We presented MPQ, a solver-side approach to post-training quantization that casts row-wise objective as posterior inference over discrete codebooks. This unified view explains classical greedy solvers as hard-decision limits, while exposing finite-temperature message-passing as a natural way to propagate codeword uncertainty across blocks before committing to a quantized solution. The benefit of postponing commitment is most visible in aggressive low-bit regimes, which are also the regimes that determine whether large language models can be deployed on commodity hardware. MPQ improves over strong solver baselines across these regimes, with the clearest gains at 2 bits. We believe message-passing inference is particularly well suited to this combinatorial problem, and will open a productive regime in the design space of low-bit quantization solvers.

Acknowledgements

This work was partially supported by NSF IIS 2347592, 2348169, DBI 2405416, CCF 2348306, CNS 2347617, RISE 2536663. The authors declare no competing interests.

References

[1] Ruibo Chen, Tianyi Xiong, Yihan Wu, Guodong Liu, Zhengmian Hu, Lichang Chen, Yanshuo Chen, Chenxi Liu, and Heng Huang. Gpt-4 vision on medical image classification–a case study on covid-19 dataset. *arXiv preprint arXiv:2310.18498*, 2023.

[2] Tianyi Xiong, Yi Ge, Ming Li, Zuolong Zhang, Pranav Kulkarni, Kaishen Wang, Qi He, Zeying Zhu, Chenxi Liu, Ruibo Chen, Tong Zheng, Yanshuo Chen, Xiyao Wang, Renrui Zhang, Wenhu Chen, and Heng Huang. Multi-crit: Benchmarking multimodal judges on pluralistic criteria-following. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 8641–8652, June 2026.

[3] Chenxi Liu, Yanshuo Chen, Ruibo Chen, Tianyi Xiong, Tong Zheng, and Heng Huang. Learning from self-debate: Preparing reasoning models for multi-agent debate. *arXiv preprint arXiv:2601.22297*, 2026.

[4] Ruibo Chen, Yihan Wu, Yanshuo Chen, Chenxi Liu, Junfeng Guo, and Heng Huang. A watermark for order-agnostic language models. In *International Conference on Learning Representations*, volume 2025, pages 4372–4388, 2025.

[5] Chenxi Liu, Tianyi Xiong, Yanshuo Chen, Ruibo Chen, Yihan Wu, Junfeng Guo, Tianyi Zhou, and Heng Huang. Modality-balancing preference optimization of large multimodal models by adversarial negative mining. *arXiv preprint arXiv:2506.08022*, 2025.

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

[7] Guangxuan Xiao, Ji Lin, Mickael Seznec, Hao Wu, Julien Demouth, and Song Han. Smoothquant: Accurate and efficient post-training quantization for large language models. In *International conference on machine learning*, pages 38087–38099. PMLR, 2023.

[8] 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. *Advances in Neural Information Processing Systems*, 37:100213–100240, 2024.

[9] 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. *arXiv preprint arXiv:2405.16406*, 2024.

[10] Jerry Chee, Yaohui Cai, Volodymyr Kuleshov, and Christopher M De Sa. Quip: 2-bit quantization of large language models with guarantees. *Advances in neural information processing systems*, 36:4396–4429, 2023.

[11] Albert Tseng, Jerry Chee, Qingyao Sun, Volodymyr Kuleshov, and Christopher De Sa. Quip#: Even better llm quantization with hadamard incoherence and lattice codebooks. *Proceedings of machine learning research*, 235:48630, 2024.

[12] Albert Tseng, Qingyao Sun, David Hou, and Christopher De. Qtip: Quantization with trellises and incoherence processing. *Advances in Neural Information Processing Systems*, 37:59597–59620, 2024.

[13] Albert Tseng, Zhaofeng Sun, and Christopher De Sa. Model-preserving adaptive rounding. *arXiv preprint arXiv:2505.22988*, 2025.

[14] Shihao Zhang, Haoyu Zhang, Ian Colbert, and Rayan Saab. Qronos: Correcting the past by shaping the future... in post-training quantization. *arXiv preprint arXiv:2505.11695*, 2025.

[15] 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.

[16] Kayhan Behdin, Ayan Acharya, Aman Gupta, Qingquan Song, Siyu Zhu, Sathiya Keerthi, and Rahul Mazumder. Quantease: Optimization-based quantization for language models. *arXiv preprint arXiv:2309.01885*, 2023.

[17] Pranav Ajit Nair and Arun Sai Suggala. Cdquant: Greedy coordinate descent for accurate llm quantization. *arXiv preprint arXiv:2406.17542*, 2024.

[18] Xinyu Wang, Ziyu Zhao, Peng Lu, Yu Gu, and Xiao-Wen Chang. Ojbkq: Objective-joint babai-klein quantization. *arXiv preprint arXiv:2602.08376*, 2026.

[19] 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. *arXiv preprint arXiv:2308.13137*, 2023.

[20] Aozhong Zhang, Naigang Wang, Yanxia Deng, Xin Li, Zi Yang, and Penghang Yin. Magr: Weight magnitude reduction for enhancing post-training quantization. *arXiv preprint arXiv:2406.00800*, 2024.

[21] 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. *Advances in Neural Information Processing Systems*, 37:5074–5121, 2024.

[22] Jiale Chen, Yalda Shabanzadeh, Elvir Crnčević, Torsten Hoefler, and Dan Alistarh. The geometry of llm quantization: Gptq as babai's nearest plane algorithm. *arXiv preprint arXiv:2507.18553*, 2025.

[23] Johann Birnick. The lattice geometry of neural network quantization–a short equivalence proof of gptq and babai's algorithm. *arXiv preprint arXiv:2508.01077*, 2025.

[24] David L Donoho, Arian Maleki, and Andrea Montanari. Message passing algorithms for compressed sensing: I. motivation and construction. In *2010 IEEE information theory workshop on information theory (ITW 2010, Cairo)*, pages 1–5. IEEE, 2010.

[25] Mohsen Bayati and Andrea Montanari. The dynamics of message passing on dense graphs, with applications to compressed sensing. *IEEE Transactions on Information Theory*, 57(2):764–785, 2011.

[26] Sundeep Rangan. Generalized approximate message passing for estimation with random linear mixing. In *2011 IEEE international symposium on information theory proceedings*, pages 2168–2172. IEEE, 2011.

[27] Sundeep Rangan, Philip Schniter, and Alyson K Fletcher. Vector approximate message passing. *IEEE Transactions on Information Theory*, 65(10):6664–6684, 2019.

[28] 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, 2023.

[29] Stephen Merity, Caiming Xiong, James Bradbury, and Richard Socher. Pointer sentinel mixture models, 2016.

[30] Peter Clark, Isaac Cowhey, Oren Etzioni, Tushar Khot, Ashish Sabharwal, Carissa Schoenick, and Oyvind Tafjord. Think you have solved question answering? try arc, the ai2 reasoning challenge. *ArXiv*, abs/1803.05457, 2018.

[31] Rowan Zellers, Ari Holtzman, Yonatan Bisk, Ali Farhadi, and Yejin Choi. Hellaswag: Can a machine really finish your sentence? In *Proceedings of the 57th Annual Meeting of the Association for Computational Linguistics*, 2019.

[32] Yonatan Bisk, Rowan Zellers, Ronan Le Bras, Jianfeng Gao, and Yejin Choi. Piqa: Reasoning about physical commonsense in natural language. In *Thirty-Fourth AAAI Conference on Artificial Intelligence*, 2020.

[33] Keisuke Sakaguchi, Ronan Le Bras, Chandra Bhagavatula, and Yejin Choi. Winogrande: An adversarial winograd schema challenge at scale. *arXiv preprint arXiv:1907.10641*, 2019.

[34] Leo Gao, Jonathan Tow, Baber Abbasi, Stella Biderman, Sid Black, Anthony DiPofi, Charles Foster, Laurence Golding, Jeffrey Hsu, Alain Le Noac'h, Haonan Li, Kyle McDonell, Niklas Muennighoff, Chris Ociepa, Jason Phang, Laria Reynolds, Hailey Schoelkopf, Aviya Skowron, Lintang Sutawika, Eric Tang, Anish Thite, Ben Wang, Kevin Wang, and Andy Zou. The language model evaluation harness, 07 2024.

[35] Hugo Touvron, Louis Martin, Kevin Stone, Peter Albert, Amjad Almahairi, Yasmine Babaei, Nikolay Bashlykov, Soumya Batra, Prajjwal Bhargava, Shruti Bhosale, et al. Llama 2: Open foundation and fine-tuned chat models. *arXiv preprint arXiv:2307.09288*, 2023.

[36] Aaron Grattafiori, Abhimanyu Dubey, Abhinav Jauhri, Abhinav Pandey, Abhishek Kadian, Ahmad Al-Dahle, Aiesha Letman, Akhil Mathur, Alan Schelten, Alex Vaughan, et al. The llama 3 herd of models. *arXiv preprint arXiv:2407.21783*, 2024.

[37] Albert Qiaochu Jiang, Alexandre Sablayrolles, Arthur Mensch, Chris Bamford, Devendra Singh Chaplot, Diego de Las Casas, Florian Bressand, Gianna Lengyel, Guillaume Lample, Lucile Saulnier, Lélio Renard Lavaud, Marie-Anne Lachaux, Pierre Stock, Teven Le Scao, Thibaut Lavril, Thomas Wang, Timothée Lacroix, and William El Sayed. Mistral 7b. *ArXiv*, abs/2310.06825, 2023.

[38] An Yang, Anfeng Li, Baosong Yang, Beichen Zhang, Binyuan Hui, Bo Zheng, Bowen Yu, Chang Gao, Chengen Huang, Chenxu Lv, et al. Qwen3 technical report. *arXiv preprint arXiv:2505.09388*, 2025.

[39] ModelCloud.ai and qubitium@modelcloud.ai. Gpt-qmodel. https://github.com/modelcloud/gptqmodel, 2024. Contact: qubitium@modelcloud.ai.

Appendix

A Additional experimental details

A.1 Pilot sweep for frozen VAMP hyperparameters

Before freezing the defaults, we ran a small pilot sweep over the three scalar VAMP hyperparameters that most directly affect convergence: the number of VAMP iterations, the damping coefficient, and the anchoring multiplier τ. Figure 5 summarizes the pilot together with the calibration-budget robustness check; hollow markers denote the selected defaults. The chosen setting $(T_{\mathrm{vamp}}, \mathrm{damping}, \tau) = (16, 0.5, 0.2)$ lies in the stable region of the sweep rather than at a brittle edge, and the Llama-3-8B 3-bit result remains stable as the calibration budget varies from 32 to 128 sequences with 3 different random seeds.

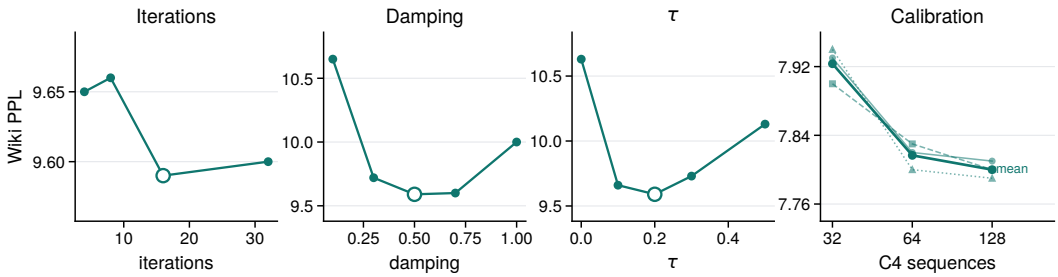

Figure 5: Pilot sweeps over VAMP iterations, damping, anchoring multiplier τ, and calibration budget for Llama-3-8B 3-bit scalar quantization.

Table 3: Full perplexity matrix for the retained seven-model evaluation. Entries are WikiText-2 / C4 PPL.

Method	Llama3.2-1B	Llama3.2-3B	Llama3-8B	Llama2-13B	Llama2-70B	Qwen3-8B	Mistral-7B
BF16	9.75 / 12.77	7.82 / 10.47	6.14 / 8.93	4.89 / 6.47	3.32 / 5.52	6.97 / 10.19	5.32 / 7.84
4-bit scalar codebook, group size 128							
GPTQ	10.63 / 13.86	8.22 / 11.08	6.54 / 9.41	4.99 / 6.57	3.43 / 5.58	7.22 / 10.58	5.42 / 7.97
GPTQ + CD	10.57 / 13.83	8.24 / 11.10	6.55 / 9.42	4.99 / 6.57	3.43 / 5.58	7.26 / 10.59	5.43 / 7.96
QuantEase	10.47 / 13.80	8.18 / 11.03	6.50 / 9.37	4.98 / 6.57	3.43 / 5.58	7.19 / 10.54	**5.41** / 7.95
OJBKQ	10.43 / 13.68	8.15 / 10.99	6.48 / 9.35	4.97 / 6.56	3.42 / 5.58	**7.16** / 10.53	**5.41 / 7.94**
MPQ	**10.38 / 13.63**	**8.12 / 10.94**	**6.46 / 9.33**	**4.96 / 6.55**	**3.41 / 5.57**	**7.16 / 10.51**	**5.41 / 7.94**
3-bit scalar codebook, group size 128							
GPTQ	15.45 / 19.25	10.56 / 13.76	8.27 / 11.59	5.46 / 7.09	3.93 / 5.89	8.19 / 11.53	5.94 / 8.56
GPTQ + CD	15.58 / 18.94	10.51 / 13.82	8.37 / 11.60	5.46 / 7.09	3.92 / 5.90	8.21 / 11.56	5.94 / 8.54
QuantEase	14.86 / 18.51	10.05 / 13.36	8.10 / 11.24	5.39 / 6.95	3.88 / 5.88	8.03 / 11.29	5.82 / 8.39
OJBKQ	14.26 / 18.00	9.93 / 13.22	7.95 / 11.08	5.40 / 6.98	3.87 / 5.86	7.89 / 11.24	5.81 / 8.36
MPQ	**13.65 / 17.42**	**9.65 / 12.87**	**7.81 / 10.88**	**5.30 / 6.89**	**3.78 / 5.80**	**7.80 / 11.13**	**5.80 / 8.35**
2-bit E8P12 codebook							
LDLQ	42.80 / 43.47	18.54 / 21.15	15.01 / 18.17	6.63 / 8.17	4.58 / 6.36	9.54 / 12.78	7.81 / 10.53
LDLQ + CD	41.79 / 42.69	18.00 / 20.66	12.76 / 15.96	6.61 / 8.13	4.59 / 6.36	9.41 / 12.76	6.98 / 9.53
QuantEase	47.31 / 46.24	18.84 / 21.17	14.78 / 17.77	6.67 / 8.19	4.70 / 6.46	10.55 / 13.80	7.48 / 10.10
OJBKQ	39.28 / 39.72	17.55 / 19.94	13.84 / 16.57	6.55 / 8.11	4.58 / 6.36	9.43 / 12.81	6.99 / 9.53
MPQ	**31.67 / 31.48**	**15.38 / 18.10**	**11.63 / 14.77**	**6.27 / 7.76**	**4.49 / 6.30**	**9.19 / 12.57**	**6.75 / 9.38**

A.2 Full perplexity matrix

Table 3 expands the main perplexity comparison to the complete seven-model set used in the paper: five Llama-family models, Qwen3-8B, and Mistral-7B. Each entry is WikiText-2 / C4 perplexity, and MPQ denotes the final CD-polished configuration. The table section headers specify the codebook and preprocessing regime.

A.3 Full zero-shot task matrix

Table 4 reports the retained zero-shot task matrix. Bold marks the best quantized result within each model, bit-width, and task; underlining marks the second-best.

Table 4: Full zero-shot task matrix for the retained models.

Model / Bits	Method	ARC-C	ARC-E	Hella	PIQA	Wino	Avg
Llama-3-8B							
BF16 reference	BF16	54.1	77.6	79.2	80.6	73.6	73.0
3-bit scalar	GPTQ	44.5	72.2	74.3	**77.4**	**70.9**	67.9
3-bit scalar	GPTQ + CD	45.1	72.0	74.5	**77.4**	69.6	67.7
3-bit scalar	QuantEase	46.5	70.5	75.0	77.3	71.2	68.1
3-bit scalar	OJBKQ	**47.9**	72.4	75.2	76.8	70.3	68.5
3-bit scalar	MPQ	47.6	**73.4**	**75.5**	76.1	70.5	**68.6**
2-bit E8P12	LDLQ	34.6	57.4	59.8	70.9	64.4	57.4
2-bit E8P12	LDLQ + CD	35.6	**60.9**	63.0	72.1	64.7	59.3
2-bit E8P12	QuantEase	30.8	53.9	59.1	69.9	62.9	55.3
2-bit E8P12	OJBKQ	35.6	59.2	64.4	71.4	65.2	59.2
2-bit E8P12	MPQ	**35.8**	58.6	**64.5**	**72.9**	**66.9**	**59.7**
Qwen3-8B							
BF16 reference	BF16	56.9	79.8	78.7	79.5	72.8	73.5
3-bit scalar	GPTQ	49.1	75.2	74.7	78.5	69.0	69.3
3-bit scalar	GPTQ + CD	51.7	76.4	75.0	78.2	69.4	70.1
3-bit scalar	QuantEase	52.8	77.4	74.9	77.8	69.2	70.4
3-bit scalar	OJBKQ	51.4	78.6	75.3	**78.8**	**71.2**	71.0
3-bit scalar	MPQ	**54.4**	**79.6**	**75.9**	78.1	70.8	**71.8**
2-bit E8P12	LDLQ	**49.0**	76.2	69.4	76.7	69.3	68.1
2-bit E8P12	LDLQ + CD	47.3	72.8	70.0	76.5	69.5	67.2
2-bit E8P12	QuantEase	40.5	66.0	67.8	75.9	67.5	63.5
2-bit E8P12	OJBKQ	45.0	72.1	**70.3**	76.2	69.1	66.5
2-bit E8P12	MPQ	48.7	**76.6**	70.0	**77.0**	**69.6**	**68.4**
Mistral-7B							
BF16 reference	BF16	54.3	80.1	80.7	82.0	74.3	74.3
3-bit scalar	GPTQ	49.6	75.4	78.0	81.0	72.1	71.2
3-bit scalar	GPTQ + CD	50.0	75.3	77.3	**81.6**	71.9	71.2
3-bit scalar	QuantEase	**51.7**	**77.9**	78.3	80.7	72.0	**72.1**
3-bit scalar	OJBKQ	50.7	76.8	**79.0**	81.1	72.0	71.9
3-bit scalar	MPQ	50.5	76.2	78.6	81.0	**73.2**	71.9
2-bit E8P12	LDLQ	41.5	70.1	**73.7**	77.6	71.3	66.8
2-bit E8P12	LDLQ + CD	43.1	70.8	73.4	78.8	**71.4**	67.5
2-bit E8P12	QuantEase	42.6	70.0	70.6	78.1	69.0	66.1
2-bit E8P12	OJBKQ	43.5	**72.2**	73.4	78.0	70.6	67.5
2-bit E8P12	MPQ	**43.8**	71.3	73.4	**79.8**	71.1	**67.9**

Table 4 continued: Full zero-shot task matrix for the retained models.

Model / Bits	Method	ARC-C	ARC-E	Hella	PIQA	Wino	Avg
Llama-2-13B							
BF16 reference	BF16	48.8	76.5	79.6	80.4	72.5	71.6
3-bit scalar	GPTQ	45.5	73.0	76.5	79.2	71.4	69.1
3-bit scalar	GPTQ + CD	46.1	73.7	76.7	79.1	71.2	69.4
3-bit scalar	QuantEase	47.5	73.3	77.5	79.5	**71.8**	69.9
3-bit scalar	OJBKQ	47.2	74.5	77.1	**80.5**	70.7	70.0
3-bit scalar	MPQ	**47.8**	73.5	**77.9**	80.1	71.0	**70.1**
2-bit E8P12	LDLQ	41.1	68.9	72.1	76.8	68.6	65.5
2-bit E8P12	LDLQ + CD	40.4	68.1	72.0	77.1	69.9	65.5
2-bit E8P12	QuantEase	41.9	69.1	71.1	**77.9**	69.1	65.8
2-bit E8P12	OJBKQ	41.6	70.0	72.3	77.0	69.2	66.0
2-bit E8P12	MPQ	**43.0**	**71.0**	**73.0**	77.1	**70.3**	**66.9**
Llama-2-70B							
BF16 reference	BF16	53.9	82.6	84.4	82.5	80.0	76.7
3-bit scalar	GPTQ	53.7	**79.1**	82.5	**82.3**	79.1	**75.3**
3-bit scalar	GPTQ + CD	53.5	77.1	82.5	82.2	**79.2**	74.9
3-bit scalar	QuantEase	53.6	78.5	83.1	81.6	78.4	75.0
3-bit scalar	OJBKQ	53.7	78.6	83.0	**82.3**	78.6	75.2
3-bit scalar	MPQ	**53.9**	78.2	**83.6**	82.1	78.6	**75.3**
2-bit E8P12	LDLQ	51.8	77.2	80.5	81.0	77.9	73.7
2-bit E8P12	LDLQ + CD	51.0	77.1	80.5	81.1	77.6	73.5
2-bit E8P12	QuantEase	53.2	78.7	80.2	80.8	**78.3**	74.2
2-bit E8P12	OJBKQ	53.4	78.2	80.4	81.4	77.4	74.2
2-bit E8P12	MPQ	**53.8**	**78.7**	**80.8**	**81.6**	78.0	**74.6**

A.4 Ablation study

Figure 6 shows the MPQ recipe variants on the middle size 8B/7B models, while Table 5 tests how AWQ affects MPQ solvers under the same RHT convention.

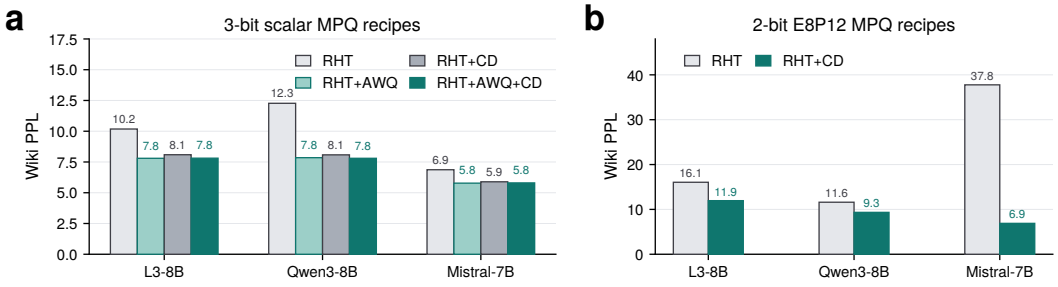

Figure 6: MPQ recipe variants on WikiText-2 for the 8B/7B models. **a** Performance comparison of 3-bit scalar preprocessing and CD choices. **b** Performance comparison of 2-bit E8P12 VAMP and CD-polished recipes. AWQ preprocessing intentionally manipulates the per-channel activation magnitude which would disrupt the lattice codebook's block-level isotropy and make quantization fail. Therefore, AWQ is not inlcuded in the 2-bit E8P12 setting for preprocessing ablation study.

A.5 Speed-quality tradeoff and memory

We measure solver-side cost on a single RTX 5090 and exclude Hessian construction, evaluation, and export from the timings. Figure 2 plots scaling against exact model parameter counts, and Table 6 reports the same solver times in minutes.

Tables 7 and 8 give the full Meta-Llama-3-8B grids corresponding to Figure 3.

Table 5: 3-bit scalar preprocessing and CD ablation on WikiText-2 / C4 perplexity. Delta denotes AWQ+RHT minus RHT-only.

Model	Method	RHT only	AWQ + RHT	Delta
Llama3.2-1B	MPQ (VAMP-only)	31.24 / 36.36	13.76 / 17.71	-17.48 / -18.65
	MPQ	14.55 / 18.33	13.65 / 17.42	-0.90 / -0.91
Llama3.2-3B	MPQ (VAMP-only)	14.76 / 18.81	9.59 / 12.92	-5.17 / -5.89
	MPQ	10.07 / 13.33	9.65 / 12.87	-0.42 / -0.46
Llama3-8B	MPQ (VAMP-only)	10.18 / 13.72	7.80 / 10.96	-2.38 / -2.76
	MPQ	8.09 / 11.24	7.81 / 10.88	-0.28 / -0.36
Qwen3-8B	MPQ (VAMP-only)	12.27 / 17.77	7.85 / 11.18	-4.42 / -6.59
	MPQ	8.08 / 11.37	7.80 / 11.13	-0.28 / -0.24
Mistral-7B	MPQ (VAMP-only)	6.87 / 9.96	5.78 / 8.37	-1.09 / -1.59
	MPQ	5.89 / 8.49	5.80 / 8.35	-0.09 / -0.15

Table 6: Solver-time scaling on a single RTX 5090.

Bits	Method	Llama3.2-1B	Llama3.2-3B	Llama3-8B	Llama2-13B
3-bit	GPTQ	0.7	1.6	2.7	4.1
	GPTQ+CD3	1.5	4.6	18.8	36.5
	QuantEase	2.8	7.2	13.8	20.8
	OJBKQ	2.2	5.4	10.1	16.5
	MPQ	1.2	4.2	17.7	37.4
2-bit	LDLQ	5.9	11.9	18.0	29.6
	LDLQ+CD3	8.0	17.4	34.6	57.2
	QuantEase	4.5	12.7	32.4	57.6
	OJBKQ	17.5	50.3	96.1	166.9
	MPQ	6.7	19.0	49.2	86.9

A.6 Theory-guided cleanup diagnostic

VAMP precision and cleanup. The VAMP cleanup diagnostic summarized in Figure 4 was run on the full Meta-Llama-3-8B model. In 3-bit scalar quantization, the median module-level final precision proxy $\hat{\gamma}_1$ is 1.77×10^5 over 224 modules, the median cleanup fraction is zero, and the global cleanup fraction is 6.44×10^{-7}. In 2-bit E8P12, the median $\hat{\gamma}_1$ drops to 1.30×10^1 over 128 fused-module checkpoints, the median cleanup fraction rises to 1.27×10^{-1}, and the global cleanup fraction is 1.29×10^{-1}. This result supports the mechanism that when VAMP is sharp, CD is nearly a no-op; when the codebook is harder, CD performs substantial local repair.

Table 7: 3-bit scalar codebook speed-quality grid on Meta-Llama-3-8B.

Method	**WikiText-2**	**C4**	**Solver time (s)**	**Memory peak (GB)**
GPTQ	8.29	11.61	161	8.21
GPTQ + CD3	8.43	11.63	1128	13.20
QuantEase	8.10	11.24	829	17.93
OJBKQ	7.95	11.03	606	15.84
MPQ-1 (VAMP-only)	8.25	11.61	154	12.61
MPQ-2 (VAMP-only)	7.96	11.20	161	13.99
MPQ-4 (VAMP-only)	7.80	11.00	199	13.37
MPQ-8 (VAMP-only)	7.77	10.95	229	13.29
MPQ-16 (VAMP-only)	7.78	10.94	276	13.06
MPQ-1 + CD1	8.20	11.39	418	13.63
MPQ-2 + CD1	8.04	11.14	425	13.52
MPQ-4 + CD1	7.87	11.02	463	13.65
MPQ-8 + CD1	7.84	10.94	493	13.44
MPQ-16 + CD1	7.78	10.91	540	13.57
MPQ-1 + CD2	8.09	11.25	683	13.09
MPQ-2 + CD2	7.97	11.11	690	13.30
MPQ-4 + CD2	7.86	11.00	728	13.22
MPQ-8 + CD2	7.80	10.95	758	14.30
MPQ-16 + CD2	7.82	10.94	805	13.67
MPQ-1 + CD3	8.09	11.24	947	13.40
MPQ-2 + CD3	7.94	11.08	954	13.50
MPQ-4 + CD3	7.88	11.00	992	13.52
MPQ-8 + CD3	7.81	10.92	1022	13.00
MPQ-16 + CD3	7.81	10.87	1069	13.55

Table 8: 2-bit E8P12 speed-quality grid on Meta-Llama-3-8B.

Method	**WikiText-2**	**C4**	**Solver time (s)**	**Memory peak (GB)**
LDLQ	15.01	18.17	1082	8.45
LDLQ + CD3	12.76	15.96	2076	17.49
QuantEase	14.78	17.77	1944	18.66
OJBKQ	13.84	16.57	5768	21.49
MPQ-1 (VAMP-only)	321.12	280.36	784	22.63
MPQ-1 + CD1	16.69	19.63	1024	22.63
MPQ-1 + CD2	14.97	18.27	1221	22.63
MPQ-1 + CD3	14.34	17.64	1418	22.63
MPQ-2 (VAMP-only)	742.85	502.59	880	22.63
MPQ-2 + CD1	17.66	21.01	1127	22.86
MPQ-2 + CD2	15.38	18.50	1324	22.86
MPQ-2 + CD3	14.58	17.87	1521	22.86
MPQ-4 (VAMP-only)	590.20	446.48	1085	22.86
MPQ-4 + CD1	15.13	18.22	1333	22.86
MPQ-4 + CD2	14.26	17.26	1530	22.86
MPQ-4 + CD3	13.98	16.93	1725	22.86
MPQ-8 (VAMP-only)	241.49	153.71	1493	22.86
MPQ-8 + CD1	12.77	16.18	1740	22.86
MPQ-8 + CD2	12.64	16.07	1937	22.86
MPQ-8 + CD3	12.37	15.77	2134	22.86
MPQ-16 (VAMP-only)	17.24	21.01	2311	22.86
MPQ-16 + CD1	11.89	14.99	2558	22.86
MPQ-16 + CD2	11.80	14.81	2754	22.86
MPQ-16 + CD3	11.63	14.77	2951	22.86

AWQ E_{lin} diagnostic. We also checked the sign of the linearized error proxy from Eq. (17) (equivalently, the integrand of Corollary E.3, Eq. (63)) on Meta-Llama-3-8B. To match the un-rotated path of Proposition 5.2 directly, we work in the RHT basis throughout: we apply RHT to the model, collect the calibration Hessian H_0 in the rotated basis from 128 C4 sequences of length 2048, and run GPTQModel AWQ scale search on the same rotated model with group size 128 to obtain $D = \mathrm{diag}(1/s)$. For each AWQ scale group we compute $E_0 = \frac{1}{n}\,\mathrm{tr}[(\beta_0 H_0 + \gamma I)^{-1}]$, $E_1 = \frac{1}{n}\,\mathrm{tr}[(\beta_0 D H_0 D + \gamma I)^{-1}]$ and the difference $\Delta E_{\text{lin}} = E_1 - E_0$ at $\beta_0 = \gamma = 1$, together with the $t = 0$ integrand. By Lemma E.2, this matches the production pipeline (AWQ before RHT) up to orthogonal similarity. AWQ decreased E_{lin} on 84/96 unique AWQ scale groups, increased it on 11/96 groups, and was numerically zero on one of the 96 groups. Figure 4b shows the distribution of relative changes. The module-group breakdown is simple, with all `down_proj` groups decreasing and the few increases concentrated in `qkv` and `gate_up`. The sign of the $t = 0$ integrand matched the observed ΔE_{lin}, after accounting for the theorem's leading minus sign, on 95/96 groups, with the only mismatch having $|\Delta E_{\text{lin}}| \approx 7.5 \times 10^{-11}$. The largest decreases occur in early `down_proj` groups, while all positive absolute changes are below 8×10^{-7}. This supports the qualitative remark that AWQ usually lowers the linear-module proxy in the AWQ+RHT setting.

This diagnostic makes the effect of CD more clearer. In the 3-bit scalar GPTQ setting, CD changes only a small fraction of final scalar assignments and gives a modest surrogate drop, matching the small and noisy PPL movement. In the 2-bit E8P12 LDLQ setting, CD performs real local repair and improves PPL substantially, but it still cannot match MPQ, suggesting that the remaining gap is a VAMP basin-selection issue rather than simply insufficient CD sweeps.

A.7 Calibration-resampling robustness

We independently sample three C4 calibration sets, each containing 128 sequences of length 2048. Within each draw, all methods use the same calibration data, preprocessing, evaluation tokens, and quantization protocol. We report the mean and sample standard deviation of final WikiText-2 and C4 perplexity. The scalar codebook uses group size 128 and AWQ+RHT preprocessing.

Table 9: Perplexity of 4-bit scalar quantization over three calibration draws (mean $\pm$ sample SD).

Model	Method	WikiText-2	C4
Llama-3.2-1B	GPTQ+CD3	10.625 ± 0.028	13.856 ± 0.017
	QuantEase	10.474 ± 0.014	13.827 ± 0.033
	OJBKQ	10.456 ± 0.023	13.672 ± 0.022
	MPQ	10.392 ± 0.010	13.629 ± 0.017
Llama-3.2-3B	GPTQ+CD3	8.264 ± 0.013	11.109 ± 0.010
	QuantEase	8.202 ± 0.006	11.038 ± 0.014
	OJBKQ	8.176 ± 0.014	11.016 ± 0.019
	MPQ	8.128 ± 0.010	10.961 ± 0.001
Llama-3-8B	GPTQ+CD3	6.557 ± 0.011	9.427 ± 0.010
	QuantEase	6.507 ± 0.024	9.369 ± 0.012
	OJBKQ	6.495 ± 0.001	9.364 ± 0.013
	MPQ	6.464 ± 0.004	9.326 ± 0.007
Llama-2-13B	GPTQ+CD3	5.005 ± 0.007	6.575 ± 0.001
	QuantEase	4.987 ± 0.008	6.573 ± 0.002
	OJBKQ	4.976 ± 0.007	6.576 ± 0.002
	MPQ	4.969 ± 0.010	6.553 ± 0.002

Table 10: Perplexity of 3-bit scalar quantization over three calibration draws (mean $\pm$ sample SD).

Model	Method	WikiText-2	C4
Llama-3.2-1B	GPTQ+CD3	15.352 ± 0.202	18.950 ± 0.191
	QuantEase	14.909 ± 0.174	18.759 ± 0.178
	OJBKQ	14.471 ± 0.260	18.346 ± 0.395
	MPQ	13.776 ± 0.091	17.502 ± 0.036
Llama-3.2-3B	GPTQ+CD3	10.575 ± 0.042	13.767 ± 0.029
	QuantEase	10.098 ± 0.135	13.467 ± 0.079
	OJBKQ	10.003 ± 0.090	13.266 ± 0.065
	MPQ	9.604 ± 0.014	12.919 ± 0.032
Llama-3-8B	GPTQ+CD3	8.429 ± 0.019	11.559 ± 0.032
	QuantEase	8.185 ± 0.097	11.327 ± 0.016
	OJBKQ	8.046 ± 0.043	11.073 ± 0.029
	MPQ	7.848 ± 0.026	10.855 ± 0.010
Llama-2-13B	GPTQ+CD3	5.463 ± 0.024	7.110 ± 0.001
	QuantEase	5.372 ± 0.015	6.980 ± 0.003
	OJBKQ	5.384 ± 0.021	7.005 ± 0.004
	MPQ	5.311 ± 0.006	6.902 ± 0.003

MPQ has the lowest mean perplexity across all settings, with larger gains at 3 bits.

A.8 D4 codebook and matched refinement

We compare LDLQ, OJBKQ ($K = 5$), and MPQ using the four-dimensional 2-bit D4 codebook and identical exact block-CD updates. Tables 11 and 12 give the objective and held-out perplexity.

Table 11: Calibration objective under matched D4 refinement. CD0 is the initial hard assignment.

Initialization	CD0	CD1	CD3
LDLQ	359.869170	347.919460	340.203384
OJBKQ ($K = 5$)	357.7222	348.1638	340.3826
MPQ	371.615070	335.443011	324.342445

Table 12: Held-out perplexity with the 2-bit D4 codebook.

Method	WikiText-2	C4
LDLQ	20.2967	24.1372
LDLQ + CD3	14.6389	17.6817
OJBKQ ($K = 5$) + CD3	15.0872	17.8354
MPQ + CD3	12.9658	15.7847

MPQ starts with the highest objective and has the lowest after the first CD sweep. After three sweeps, it also has the lowest WikiText-2 and C4 perplexity.

A.9 Matched solver budgets and extended refinement

Table 13 compares MPQ and LDLQ at similar solver-time budgets on Llama-3-8B with E8P12. Timings follow the protocol in Section 6.4.

At the higher budget, MPQ achieves lower perplexity than LDLQ with slightly less solver time. Its advantage over LDLQ persists at 20 CD sweeps (Table 14).

Table 13: Solver time and WikiText-2 perplexity for two offline compute budgets.

Method	Time (s)	WikiText-2
LDLQ + CD3	2076	12.756
MPQ ($T_{\mathrm{vamp}} = 8$) + CD3	2134	12.370
LDLQ + CD6	3057	12.728
MPQ ($T_{\mathrm{vamp}} = 16$) + CD3	2951	11.630

Table 14: Perplexity after extended CD refinement on Llama-3-8B with E8P12.

Initialization	CD sweeps	WikiText-2	C4
LDLQ	3	12.756	16.034
MPQ	3	11.753	14.818
LDLQ	20	12.651	15.943
MPQ	20	11.669	14.728

B Implementation

This appendix collects the exact update formulas and module identities referenced in Section 4, and concludes with a Python-style pseudocode listing of our reference implementation.

B.1 Extrinsic-message updates, initialization, and β update

Extrinsic updates. Let $(\hat{x}_1^t, \alpha_1^t)$ denote the denoiser output and sensitivity. The outgoing precision and mean are

$$\eta_1^t = \gamma_1^t/\alpha_1^t, \qquad \gamma_2^t = \eta_1^t - \gamma_1^t, \qquad r_2^t = \frac{\eta_1^t \hat{x}_1^t - \gamma_1^t r_1^t}{\gamma_2^t}. \tag{19}$$

Similarly, given the linear-module output $(\hat{x}_2^t, \alpha_2^t)$,

$$\eta_2^t = \gamma_2^t/\alpha_2^t, \qquad \gamma_1^{t+1} = \eta_2^t - \gamma_2^t, \qquad r_1^{t+1} = \frac{\eta_2^t \hat{x}_2^t - \gamma_2^t r_2^t}{\gamma_1^{t+1}}. \tag{20}$$

After each extrinsic update we damp both means and precisions geometrically with a single global damping coefficient $\rho_{\mathrm{damp}} \in (0, 1]$ and clip precisions to a fixed positive interval $[\gamma_{\min}, \gamma_{\max}]$ to avoid degenerate messages.

Initialization. We initialize $r_1^0 \leftarrow w_{\mathrm{rtn}}$, where w_{rtn} denotes the round-to-nearest rounding of w, and set the initial precision per row by

$$\gamma_1^0 = \frac{1}{\mathrm{MSE}(w, w_{\mathrm{rtn}})}, \qquad \mathrm{MSE}(w, w_{\mathrm{rtn}}) := \frac{1}{n}\|w - w_{\mathrm{rtn}}\|_2^2, \tag{21}$$

clipped below by a small numerical floor. This per-row scaling adapts γ_1^0 to the difficulty of the row and removes one continuous hyperparameter. The observation-noise precision is initialized by $\beta^0 = 1/\overline{\mathrm{MSE}}$ with $\overline{\mathrm{MSE}}$ averaged over the row batch.

Online M-step for β. We treat the inverse temperature β in the Gibbs measure of Eq. (2) as the observation-noise precision of the linear module (Eq. (31)) and update it after each linear-module step by the standard Gaussian-likelihood M-step under that posterior. In rank space, with $u_\ell := \tilde{y}_\ell - s_\ell v_\ell^\top r_2^t$ the cavity residual along the ℓ-th singular direction and $d_\ell := \beta^t s_\ell^2 + \gamma_2^t$, the update is

$$\frac{1}{\beta^{t+1}} \;\leftarrow\; \frac{1}{r}\sum_{\ell=1}^{r}\left[\left(\frac{\gamma_2^t}{d_\ell}\right)^2 u_\ell^2 + \frac{s_\ell^2}{d_\ell}\right], \tag{22}$$

averaged over the row batch when batched, then lightly damped and clipped to a fixed numerical range. The two summands in Eq. (22) are the squared posterior-mean residual and the posterior-trace correction, respectively. Removing β as a free hyperparameter gave more stable behavior than fixing it to a constant; the main tables therefore use ρ_{damp} as the only continuous solver hyperparameter.

B.2 Exact denoiser sensitivity

For one block, define

$$q(c \mid r, \gamma) = \frac{\rho(c) \exp\left(-\frac{\gamma}{2}\|c - r\|_2^2\right)}{\sum_{c' \in \mathcal{C}} \rho(c') \exp\left(-\frac{\gamma}{2}\|c' - r\|_2^2\right)}. \tag{23}$$

The mean and covariance are

$$\hat{x}(r) = \sum_c c\, q(c \mid r, \gamma), \qquad \Sigma(r) = \sum_c (c - \hat{x}(r))(c - \hat{x}(r))^\top q(c \mid r, \gamma). \tag{24}$$

Proposition B.1 (Discrete denoiser Jacobian)**.** *For every finite codebook and finite $\gamma > 0$,*

$$\frac{\partial \hat{x}(r)}{\partial r} = \gamma\, \Sigma(r). \tag{25}$$

Consequently, the average sensitivity used by MPQ is exactly

$$\alpha_1^t = \frac{\gamma_1^t}{n} \sum_{b=1}^{B} \operatorname{tr}\, \Sigma_{1,b}^t. \tag{26}$$

Proof. Differentiate the normalized discrete posterior. Writing $q_c := q(c \mid r, \gamma)$ and $\hat{x} := \hat{x}(r)$,

$$\frac{\partial q_c}{\partial r} = \gamma q_c (c - \hat{x}). \tag{27}$$

Therefore

$$\frac{\partial \hat{x}}{\partial r} = \sum_c c \left(\frac{\partial q_c}{\partial r}\right)^\top = \gamma \sum_c q_c\, c(c - \hat{x})^\top \tag{28}$$

$$= \gamma \sum_c q_c\, (c - \hat{x})(c - \hat{x})^\top + \gamma\, \hat{x} \left(\sum_c q_c (c - \hat{x})\right)^\top \tag{29}$$

$$= \gamma \sum_c q_c (c - \hat{x})(c - \hat{x})^\top = \gamma\, \Sigma(r), \tag{30}$$

where the second line writes $c = (c - \hat{x}) + \hat{x}$ inside the sum, and the third line uses $\sum_c q_c (c - \hat{x}) = 0$. Taking traces blockwise and averaging yields Eq. (26). □

B.3 Exact linear-module sensitivity

Proposition B.2 (Linear sensitivity identity)**.** *For the linear module*

$$\hat{x}_2 = (\beta H + \gamma_2 I)^{-1} (\beta g + \gamma_2 r_2), \tag{31}$$

its Jacobian with respect to r_2 is

$$\frac{\partial \hat{x}_2}{\partial r_2} = \gamma_2 (\beta H + \gamma_2 I)^{-1}, \tag{32}$$

and hence the average sensitivity is

$$\alpha_2 = \frac{1}{n} \operatorname{tr}\left[\gamma_2 (\beta H + \gamma_2 I)^{-1}\right]. \tag{33}$$

Proof. The map $r_2 \mapsto \hat{x}_2$ is affine, so its Jacobian is the coefficient of r_2, namely $\gamma_2 (\beta H + \gamma_2 I)^{-1}$. Averaging the diagonal entries gives Eq. (33). □

B.4 Block coordinate descent search as Mahalanobis nearest-codeword search

Let $u = \tilde{y} - Tx$ be the current residual. For block b, define the local Gram matrix $G_b := T_b^\top T_b$ and the local score $h_b := T_b^\top u$.

Proposition B.3 (Equivalent blockwise Mahalanobis search)**.** *Up to an additive constant independent of c, the block objective is*

$$F_b(c) = \frac{1}{2}(c - \mu_b)^\top G_b (c - \mu_b) + \text{const}, \qquad \mu_b := x_b + G_b^{-1} h_b, \tag{34}$$

whenever G_b is nonsingular. Thus exact block search is equivalent to nearest-codeword search in the Mahalanobis metric induced by G_b. If G_b is singular, the algorithm still minimizes the quadratic form in Eq. (11) *directly. Note that $h_b = T_b^\top u \in \text{range}(T_b^\top) = \text{range}(T_b^\top T_b) = \text{range}(G_b)$, so $G_b^+ h_b$ is well defined; equivalently, one may use the Moore–Penrose pseudoinverse with $\mu_b := x_b + G_b^+ h_b$ and the induced positive-semidefinite seminorm.*

Proof. Expand Eq. (11):

$$F_b(c) = \frac{1}{2}\|u\|_2^2 - u^\top T_b (c - x_b) + \frac{1}{2}(c - x_b)^\top G_b (c - x_b) \tag{35}$$

$$= \frac{1}{2}(c - x_b)^\top G_b (c - x_b) - h_b^\top (c - x_b) + \text{const}. \tag{36}$$

Completing the square around $\mu_b = x_b + G_b^{-1} h_b$ yields Eq. (34). □

Proposition B.3 makes explicit why initialization still matters even though the second stage is exact within each block. The search metric itself depends on the current residual, so the later refinement remains local.

B.5 Reference implementation

The pseudocode below gives Python-style pseudocode for one row block of MPQ. Compared with Algorithm 1, it makes explicit the precision retry/clipping safeguards on α_1 and γ_2, and a best-iterate selection rule (initialized to the RTN baseline) that passes the hard MAP projection with the smallest rank-space surrogate error to CD initialization.

```
def mpq_solve_row_batch(W, U, Sigma_A, V, codebooks, sigma, tau,
                        T_vamp, S_CD, damping):
    # ---- Shape definitions ----------------------------------------
    # W: (nrow, n) full-precision rows (nrow row-batch size, n features)
    # U, Sigma_A, V: compact SVD of A = U * diag(Sigma_A) * V^T,
    #   with Sigma_A of shape (r,) for r = rank(A)
    # codebooks[b]: (M_b, d) codeword matrix for block b
    # sigma, tau: parameters of rho_b in Eq. (2)
    nrow, n    = W.shape
    r          = Sigma_A.shape[0]
    d          = codebooks[0].shape[1]
    num_blocks = n // d                          # equals B in the paper
    # b_slice(b) := slice(b*d, (b+1)*d)          # zero-based block slice

    # ---- Stage I: finite-temperature VAMP in rank space ----
    x_rtn = round_to_nearest(W, codebooks)
    y_tilde = W @ V * Sigma_A   # (nrow, r); row j equals U^T A w_j
    s = Sigma_A # (r,)
    r1 = x_rtn.copy()                                    # (nrow, n)
    mse_init = ((W - x_rtn) ** 2).mean(axis=1)           # (nrow,)
    gamma1 = 1 / mse_init.clip(min=1e-12)               # (nrow,) per-row init
    beta = 1.0 / mse_init.mean().clip(min=1e-12)         # scalar EM init
    r2_prev, gamma2_prev = None, None
```

```
for t in range(T_vamp):
    # ---- Denoiser (Eqs. (4)-(6)); retry gamma1 on degenerate alpha1 ----
    for _ in range(MAX_RETRIES):
        q, x1, Sigma1 = denoiser_posterior(
            r1, gamma1, codebooks,
            prior_center=W, sigma=sigma, tau=tau)
        alpha1 = gamma1 * trace_block_sum(Sigma1) / n      # Eq. (6)
        if alpha_in_safe_range(alpha1):
            break
        gamma1 = retry_rescale(gamma1, alpha1)             # multiplicative
    alpha1 = clip_alpha(alpha1)

    # ---- Extrinsic 1 -> 2 (Eqs. (19)-(20)); damping, clipping ----
    eta1   = gamma1 / alpha1
    gamma2 = clip_precision(eta1 - gamma1)
    r2     = (eta1 * x1 - gamma1 * r1) / gamma2
    if r2_prev is not None and damping < 1.0:
        gamma2 = log_damp(gamma2_prev, gamma2, damping)
        r2     = (1 - damping) * r2_prev + damping * r2

    # ---- Linear module in rank space (Eqs. (7)-(8)) ----
    denom      = beta * s ** 2 + gamma2[:, None]           # (nrow, r)
    residual_u = y_tilde - (r2 @ V) * s                    # (nrow, r)
    x2         = r2 + (residual_u * (beta * s) / denom) @ V.T
    alpha2_raw = ((1 - r / n) +
                  (gamma2 / n) * (1.0 / denom).sum(axis=1))
    alpha2     = clip_alpha(alpha2_raw)

    # ---- EM update of beta (App. B.1) ----
    beta = em_update_beta(residual_u, gamma2, s, beta,
                          num_obs=r, damping=BETA_EM_DAMPING)

    # ---- Extrinsic 2 -> 1 (Eqs. (21)-(22)); damping ----
    eta2       = gamma2 / alpha2
    gamma1_new = clip_precision(eta2 - gamma2)
    r1_new     = (eta2 * x2 - gamma2 * r2) / gamma1_new
    if damping < 1.0:
        gamma1 = log_damp(gamma1, gamma1_new, damping)
        r1     = (1 - damping) * r1 + damping * r1_new
    else:
        gamma1, r1 = gamma1_new, r1_new
    r2_prev, gamma2_prev = r2, gamma2

# ---- MAP projection ----
q_final, _, _ = denoiser_posterior(
    r1, gamma1, codebooks,
    prior_center=W, sigma=sigma, tau=tau
)
x = blockwise_argmax(q_final, codebooks).copy()

# ---- Stage II: exact block coordinate descent (Eqs. (11)-(12)) ----
# In rank space T = diag(Sigma_A) . V^T;
# column block b is s[:, None] * V[b_slice(b)].T.
u = y_tilde - (x @ V) * s                                  # residual
for sweep in range(S_CD):
    changed = False
    for b in range(num_blocks):
        x_b_old = x[:, b_slice(b)]
        T_b     = s[:, None] * V[b_slice(b)].T             # (r, d)
```

```
            x_b_new = argmin_F_b(u, T_b=T_b,
                                 x_b_old=x_b_old,
                                 codebook=codebooks[b])        # Eq. (12)
            if not equal(x_b_new, x_b_old):
                u -= (x_b_new - x_b_old) @ T_b.T               # update residual
                x[:, b_slice(b)] = x_b_new
                changed = True
        if not changed:
            break
    return x
```

Helper contracts. The pseudocode above uses a small set of helper routines. Their contracts are as follows; all are deterministic, and their references make the listing self-contained.

- `round_to_nearest(W, codebooks)` returns, for each row of W and each block b, the codeword in $\mathcal{C}_b$ closest in ℓ_2 to the corresponding block of the row.
- `denoiser_posterior(r1, gamma1, codebooks, prior_center=W, sigma, tau)` evaluates Eqs. (23)–(24) blockwise with ρ_b defined as in Eq. (2); it returns the blockwise posterior q, mean $\hat{x}_1$, and covariance Σ_1.
- `trace_block_sum(Sigma1)` computes $\sum_{b=1}^{B} \operatorname{tr} \Sigma_{1,b}$, a row-wise scalar.
- `blockwise_argmax(q, codebooks)` returns $\hat{x}$ with $\hat{x}_b = \arg\max_{c \in \mathcal{C}_b} q_b(c)$, used for MAP projection (Eq. (9)) and for the best-iterate diagnostic.
- `argmin_F_b(u, T_b, x_b_old, codebook)` solves Eq. (12) by enumeration over `codebook`, given the current rank-space residual u and the rank-space block operator T_b.
- `em_update_beta(residual_u, gamma2, s, beta, num_obs)` returns the EM-style update of β described in Eq. (22).
- `clip_alpha(a)` clips into $[\varepsilon, 1-\varepsilon]$ for a small fixed ε; `clip_precision(g)` clips into $[\gamma_{\min}, \gamma_{\max}]$ for fixed positive constants.
- `alpha_in_safe_range(alpha1)` tests whether $\alpha_1 \in [\varepsilon, 1-\varepsilon]$ for the same ε.
- `retry_rescale(gamma1, alpha1)` multiplies γ_1 by a fixed factor > 1 when $\alpha_1 \geq 1-\varepsilon$ and by its reciprocal when $\alpha_1 \leq \varepsilon$, leaving γ_1 otherwise unchanged.
- `log_damp(prev, new, damping)` returns $\exp\big((1 - \texttt{damping}) \log \texttt{prev} + \texttt{damping} \log \texttt{new}\big)$, the geometric (log-space) damping used for precisions.
- `b_slice(b)` returns the (Python, zero-based) index slice $[bd, (b+1)d)$ for the b-th block of width d; `equal` returns true when its two arguments are identical, where blocks may be represented either as codeword vectors or by their stored integer codeword indices.

Implementation notes. First, the per-row choice of γ_1^0 via $1/\mathrm{MSE}(w, w_{\mathrm{rtn}})$ removes one continuous hyperparameter. State evolution is locally insensitive to this choice, but per-row scaling avoids pathologically small initial precisions on easy rows. Second, the precision retries on α_1 and the precision clipping on γ_2, γ_1^{t+1} guard against the rare degenerate cavities that finite discrete-denoiser sensitivities can produce. These safeguards activate on a small fraction of iterations in our runs and do not change the qualitative behavior of the recursion.

C Limitations

Asymptotic theory. The analysis in Section 5 takes the unanchored $\tau \to \infty$ limit and adopts an idealized right-orthogonally invariant surrogate. The MPQ algorithm itself runs at finite τ on Hessians that are RHT-conditioned but not exactly right-orthogonally invariant in the asymptotic sense. We therefore treat Theorem 5.1 and Proposition 5.2 as qualitative guidance rather than tight characterizations of the deployed algorithm. A finite-τ and finite-dimensional analysis remains open.

Codebook scope. Our experiments cover uniform scalar codebooks at 3 and 4 bits and E8P12 and D4 lattice codebooks at 2 bits. Adapting MPQ to learned vector codebooks (AQLM) or trellis-coded codebooks (QTIP) is straightforward in principle, since the message-passing view in Section 3 is codebook-agnostic, but it requires codebook-specific extensions to the discrete denoiser that we do not evaluate here.

D Broader impact

This work proposes an improved solver-side algorithm for post-training weight quantization of large language models. Our contribution sits within the foundational research category, and we view its broader societal effects as inheriting from those of the underlying language models rather than introducing new categories of risk.

Potential positive impacts include lower memory and compute requirements for deploying large language models. This can broaden access to language modeling technology in resource-constrained settings, including smaller research labs, edge devices, and regions with limited cloud infrastructure, and can reduce the energy footprint of inference at scale.

Potential negative impacts derive from the same lowering of deployment cost. Quantization techniques that make large language models cheaper to serve can also accelerate the deployment of language models in applications with harmful intent, such as disinformation and automated abuse. We do not see a direct or unique path from MPQ specifically to such harms beyond what is shared by the broader class of weight quantization methods and by language model deployment in general. The present submission releases an offline quantization solver rather than pre-trained models, fine-tuned weights, or training data.

E Detailed proofs

This appendix provides full proofs for the two main theorems stated in Section 5 and records the supporting results. Exact implementation identities used below are collected in Appendix B.

E.1 Auxiliary specialization of the right-orthogonally invariant (ROI) state evolution

We restate the right-orthogonally invariant surrogate here for completeness. We analyze one row-wise PTQ subproblem through

$$y = Ax^\star + \xi, \qquad \xi \sim \mathcal{N}(0, \beta_0^{-1} I_m), \tag{37}$$

where $x^\star = [C_1^\star; \dots; C_B^\star]$ has fixed block size d and i.i.d. blocks from the uniform prior $\pi = \mathrm{Unif}(\mathcal{C})$ on $\mathcal{C} \subset \mathbb{R}^d$. This is the unanchored $\tau \to \infty$ limit of the reference measure in Eq. (2); finite τ in the algorithm is treated as an anchoring regularizer rather than as part of this surrogate. The matrix $A \in \mathbb{R}^{m \times n}$ is assumed ROI with limiting spectral law ν for $H = A^\top A$. For a precision $\gamma > 0$, the block-discrete denoiser is

$$g_d(r; \gamma) = \sum_{c \in \mathcal{C}} c\, q(c \mid r, \gamma), \qquad q(c \mid r, \gamma) \propto \exp\Big(-\frac{\gamma}{2}\|r - c\|_2^2\Big), \tag{38}$$

and the denoiser and linear error maps are

$$E_d(\gamma) = \frac{1}{d}\,\mathbb{E}\Big[\|g_d(C + \gamma^{-1/2} G; \gamma) - C\|_2^2\Big], \qquad E_{\mathrm{lin}}(\gamma) = \int \frac{1}{\beta_0 \lambda + \gamma}\, d\nu(\lambda), \tag{39}$$

with $C \sim \mathrm{Unif}(\mathcal{C})$ and $G \sim \mathcal{N}(0, I_d)$. The integral expression for E_{lin} is the large-system counterpart of the finite-n trace in Eq. (16). The corresponding extrinsic-precision maps are $\Psi_d(\gamma) := E_d(\gamma)^{-1} - \gamma$ and $\Psi_{\mathrm{lin}}(\gamma) := E_{\mathrm{lin}}(\gamma)^{-1} - \gamma$.

Proposition E.1 (SE specialization to the MPQ denoiser)**.** *Under the surrogate model* (37) *and the standard fixed-size block-separable extension of the right-orthogonally invariant assumptions of [27, Theorem 1], the message-passing stage precisions obey*

$$\gamma_2^t = \Psi_d(\gamma_1^t), \qquad \gamma_1^{t+1} = \Psi_{\mathrm{lin}}(\gamma_2^t). \tag{40}$$

Equivalently, each block sees the effective AWGN channel

$$R_b^t = C_b^\star + (\gamma_1^t)^{-1/2} G_b, \qquad G_b \sim \mathcal{N}(0, I_d). \tag{41}$$

Proof. This follows by applying the standard fixed-size block lifting of the VAMP state-evolution result of [27, Theorem 1]. On the block-lifted system, the MPQ denoiser is separable across fixed-size blocks and smooth for every finite γ because the codebook is finite. By Proposition B.1 in Appendix B, the matched denoiser sensitivity is

$$\alpha_1 = \frac{\gamma_1}{d}\,\mathbb{E}\ \mathrm{tr}\ \mathrm{Cov}(C \mid R) = \gamma_1 E_d(\gamma_1). \tag{42}$$

Likewise, Proposition B.2 yields the matched linear sensitivity

$$\alpha_2 = \gamma_2 E_{\mathrm{lin}}(\gamma_2). \tag{43}$$

Substituting Eqs. (42)–(43) into the standard VAMP extrinsic precision updates,

$$\eta_i = \gamma_i/\alpha_i, \qquad \bar{\gamma}_i = \eta_i - \gamma_i, \tag{44}$$

produces

$$\bar{\gamma}_1 = E_d(\gamma_1)^{-1} - \gamma_1, \qquad \bar{\gamma}_2 = E_{\mathrm{lin}}(\gamma_2)^{-1} - \gamma_2, \tag{45}$$

which is exactly Eq. (40). The effective-channel representation follows from the scalar-equivalent model delivered by the VAMP theorem, and local convergence holds when the recursion is initialized in the basin of the selected attractive fixed point. □

E.2 Proof of Theorem 5.1

Proof. For each competitor $j \neq j^\star$, under $R = C^\star + \eta$ and the uniform prior, we have

$$s_j(R) - s_{j^\star}(R) = -\Delta_j + \gamma_1^\star \langle \eta, \delta_j \rangle, \qquad \Delta_j := \frac{\gamma_1^\star}{2}\|\delta_j\|_2^2. \tag{46}$$

Since $\langle \eta, \delta_j \rangle \sim \mathcal{N}\big(0, (\gamma_1^\star)^{-1}\|\delta_j\|_2^2\big)$, the score difference is Gaussian with mean $-\Delta_j$ and variance $\gamma_1^\star\|\delta_j\|_2^2$. Therefore

$$\mathbb{P}\big(s_j(R) \geq s_{j^\star}(R)\big) = Q_{\mathrm{G}}\left(\frac{\sqrt{\gamma_1^\star}\|\delta_j\|_2}{2}\right), \tag{47}$$

where $Q_{\mathrm{G}}(t) = \mathbb{P}(Z \geq t)$ for $Z \sim \mathcal{N}(0,1)$. Applying the Gaussian tail bound $Q_{\mathrm{G}}(t) \leq e^{-t^2/2}$ for $t \geq 0$ (the argument $\sqrt{\gamma_1^\star}\|\delta_j\|_2/2$ is nonnegative since $\delta_j \neq 0$ for $j \neq j^\star$ and $\gamma_1^\star > 0$) gives

$$\mathbb{P}\big(s_j(R) \geq s_{j^\star}(R)\big) \leq \exp\!\left(-\frac{\gamma_1^\star\|\delta_j\|_2^2}{8}\right). \tag{48}$$

The MAP decoder fails only if at least one competitor outranks the true codeword, so a union bound over $j \neq j^\star$ yields Eq. (15).

For the expected number of incorrect blocks under the i.i.d. block surrogate, let E_b be the event that block b is decoded incorrectly by the MAP projection. Conditional on a fixed true codeword, the same bound applies to each block, and summing over b gives the factor B. Averaging the conditional bound over the uniform draw of $C_b^\star$ gives the unconditional bound. □

E.3 Proof of Proposition 5.2

We first record the orthogonal-similarity invariance that justifies analyzing the unrotated path $H_t = D_t H_0 D_t$ in the AWQ+RHT pipeline.

Lemma E.2 (Orthogonal invariance of the linear module)**.** *For any orthogonal matrix O and any symmetric $H \succeq 0$,*

$$\frac{1}{n}\,\mathrm{tr}\big[(\beta_0 O^\top H O + \gamma I)^{-1}\big] = \frac{1}{n}\,\mathrm{tr}\big[(\beta_0 H + \gamma I)^{-1}\big]. \tag{49}$$

Consequently, if

$$H_{\mathrm{raw}} = O^\top H_0 O, \qquad H_{\mathrm{prep}} = O^\top D H_0 D O, \tag{50}$$

as in the AWQ+RHT pipeline, then

$$E_{\mathrm{lin}}(H_{\mathrm{prep}}, \gamma) - E_{\mathrm{lin}}(H_{\mathrm{raw}}, \gamma) = E_{\mathrm{lin}}(D H_0 D, \gamma) - E_{\mathrm{lin}}(H_0, \gamma). \tag{51}$$

Proof. Since $(O^\top MO)^{-1} = O^\top M^{-1} O$ for any invertible matrix M, we have

$$(\beta_0 O^\top H O + \gamma I)^{-1} = O^\top (\beta_0 H + \gamma I)^{-1} O. \tag{52}$$

Taking traces and using cyclicity gives Eq. (49). □

Proof of Theorem 5.2. By Lemma E.2, the AWQ+RHT comparison reduces to the unrotated path $H_t = D_t H_0 D_t$, which we now analyze. Define

$$Q_t(\gamma) := \beta_0 H_t + \gamma I, \qquad E_{\mathrm{lin}}(H_t, \gamma) = \frac{1}{n} \operatorname{tr}\big[Q_t(\gamma)^{-1}\big]. \tag{53}$$

Since $D_t = e^{tL}$, differentiating $H_t = D_t H_0 D_t$ gives

$$\dot{H}_t = L H_t + H_t L. \tag{54}$$

Hence

$$\dot{Q}_t = \beta_0 (L H_t + H_t L). \tag{55}$$

Using the inverse-derivative identity $\frac{d}{dt} Q_t^{-1} = -Q_t^{-1} \dot{Q}_t Q_t^{-1}$, we obtain

$$\frac{d}{dt} E_{\mathrm{lin}}(H_t, \gamma) = \frac{1}{n} \frac{d}{dt} \operatorname{tr}\big[Q_t^{-1}\big] \tag{56}$$

$$= -\frac{1}{n} \operatorname{tr}\big[Q_t^{-1} \dot{Q}_t Q_t^{-1}\big] \tag{57}$$

$$= -\frac{\beta_0}{n} \operatorname{tr}\big[Q_t^{-1} (L H_t + H_t L) Q_t^{-1}\big]. \tag{58}$$

Now Q_t is a polynomial in H_t, so Q_t^{-1} commutes with H_t; we use this commutation below. Therefore

$$\operatorname{tr}\big[Q_t^{-1} L H_t Q_t^{-1}\big] = \operatorname{tr}\big[L H_t Q_t^{-2}\big], \tag{59}$$

$$\operatorname{tr}\big[Q_t^{-1} H_t L Q_t^{-1}\big] = \operatorname{tr}\big[L Q_t^{-1} H_t Q_t^{-1}\big] = \operatorname{tr}\big[L H_t Q_t^{-2}\big]. \tag{60}$$

Substituting into Eq. (58) yields Eq. (17),

$$\frac{d}{dt} E_{\mathrm{lin}}(H_t, \gamma) = -\frac{2\beta_0}{n} \operatorname{tr}\big[L H_t (\beta_0 H_t + \gamma I)^{-2}\big]. \tag{61}$$

Integrating from $t = 0$ to $t = 1$ gives Eq. (18). □

For interpretation, it is convenient to rewrite the trace coordinatewise.

Corollary E.3 (Coordinate form of the resolvent response)**.** *Let*

$$\kappa_i(t, \gamma) := \big[H_t (\beta_0 H_t + \gamma I)^{-2}\big]_{ii}. \tag{62}$$

Then $\kappa_i(t, \gamma) \geq 0$ for every i, t, γ, and

$$E_{\mathrm{lin}}(H_1, \gamma) - E_{\mathrm{lin}}(H_0, \gamma) = -\frac{2\beta_0}{n} \sum_{i=1}^{n} \ell_i \int_0^1 \kappa_i(t, \gamma)\, dt. \tag{63}$$

Proof. Because $L = \operatorname{diag}(\ell_1, \ldots, \ell_n)$ is diagonal,

$$\operatorname{tr}\big[L H_t (\beta_0 H_t + \gamma I)^{-2}\big] = \sum_{i=1}^{n} \ell_i \big[H_t (\beta_0 H_t + \gamma I)^{-2}\big]_{ii}, \tag{64}$$

which gives Eq. (63). To see nonnegativity, diagonalize $H_t = U_t \Lambda_t U_t^\top$ with $\Lambda_t = \operatorname{diag}(\lambda_1, \ldots, \lambda_n)$, $\lambda_j \geq 0$. Then

$$H_t (\beta_0 H_t + \gamma I)^{-2} = U_t \operatorname{diag}\left(\frac{\lambda_j}{(\beta_0 \lambda_j + \gamma)^2}\right) U_t^\top. \tag{65}$$

This matrix is positive semidefinite, so every diagonal entry, in particular $\kappa_i(t, \gamma)$, is nonnegative. □

F Existing Assets

Datasets.

- WikiText-2 [29], CC-BY-SA 3.0;
- C4 [28], ODC-BY 1.0 (released through the AllenAI mirror);
- ARC-Challenge and ARC-Easy [30], CC-BY-SA 4.0;
- HellaSwag [31], MIT License;
- PIQA [32], Academic Free License v3.0;
- Winogrande [33], CC-BY 4.0.

Models.

- Llama-2-13B and Llama-2-70B [35], Llama 2 Community License Agreement;
- Llama-3-8B, Llama-3.2-1B, and Llama-3.2-3B [36], Meta Llama 3 Community License;
- Qwen3-8B [38], Apache License 2.0;
- Mistral-7B-v0.3 [37], Apache License 2.0.

Tooling.

- lm-evaluation-harness [34], MIT License.
- GPTQModel [39], Apache License 2.0;
- QuIP# [11], GPL-3.0 license.

NeurIPS Paper Checklist

1. **Claims**

   Question: Do the main claims made in the abstract and introduction accurately reflect the paper's contributions and scope?

   Answer: [Yes]

   Justification: The abstract and Section 1 state the message-passing view of row-wise PTQ, the MPQ algorithm, and empirical improvements over standard hard-decision baselines. Section 6 reports results matching these claims across seven dense models and three bit budgets.

   Guidelines:

   - The answer [N/A] means that the abstract and introduction do not include the claims made in the paper.
   - The abstract and/or introduction should clearly state the claims made, including the contributions made in the paper and important assumptions and limitations. A [No] or [N/A] answer to this question will not be perceived well by the reviewers.
   - The claims made should match theoretical and experimental results, and reflect how much the results can be expected to generalize to other settings.
   - It is fine to include aspirational goals as motivation as long as it is clear that these goals are not attained by the paper.

2. **Limitations**

   Question: Does the paper discuss the limitations of the work performed by the authors?

   Answer: [Yes]

   Justification: A dedicated Limitations section in Appendix C discusses the asymptotic-only theory and the codebook scope of the empirical evaluation.

   Guidelines:

   - The answer [N/A] means that the paper has no limitation while the answer [No] means that the paper has limitations, but those are not discussed in the paper.
   - The authors are encouraged to create a separate "Limitations" section in their paper.
   - The paper should point out any strong assumptions and how robust the results are to violations of these assumptions (e.g., independence assumptions, noiseless settings, model well-specification, asymptotic approximations only holding locally). The authors should reflect on how these assumptions might be violated in practice and what the implications would be.
   - The authors should reflect on the scope of the claims made, e.g., if the approach was only tested on a few datasets or with a few runs. In general, empirical results often depend on implicit assumptions, which should be articulated.
   - The authors should reflect on the factors that influence the performance of the approach. For example, a facial recognition algorithm may perform poorly when image resolution is low or images are taken in low lighting. Or a speech-to-text system might not be used reliably to provide closed captions for online lectures because it fails to handle technical jargon.
   - The authors should discuss the computational efficiency of the proposed algorithms and how they scale with dataset size.
   - If applicable, the authors should discuss possible limitations of their approach to address problems of privacy and fairness.
   - While the authors might fear that complete honesty about limitations might be used by reviewers as grounds for rejection, a worse outcome might be that reviewers discover limitations that aren't acknowledged in the paper. The authors should use their best judgment and recognize that individual actions in favor of transparency play an important role in developing norms that preserve the integrity of the community. Reviewers will be specifically instructed to not penalize honesty concerning limitations.

3. **Theory assumptions and proofs**

Question: For each theoretical result, does the paper provide the full set of assumptions and a complete (and correct) proof?

Answer: [Yes]

Justification: Theorem 5.1 and Proposition 5.2 state assumptions explicitly. Proof sketches appear in the main text; complete proofs and the auxiliary state-evolution specialization are in Appendix E.

Guidelines:

- The answer [N/A] means that the paper does not include theoretical results.
- All the theorems, formulas, and proofs in the paper should be numbered and cross-referenced.
- All assumptions should be clearly stated or referenced in the statement of any theorems.
- The proofs can either appear in the main paper or the supplemental material, but if they appear in the supplemental material, the authors are encouraged to provide a short proof sketch to provide intuition.
- Inversely, any informal proof provided in the core of the paper should be complemented by formal proofs provided in appendix or supplemental material.
- Theorems and Lemmas that the proof relies upon should be properly referenced.

4. **Experimental result reproducibility**

Question: Does the paper fully disclose all the information needed to reproduce the main experimental results of the paper to the extent that it affects the main claims and/or conclusions of the paper (regardless of whether the code and data are provided or not)?

Answer: [Yes]

Justification: Section 6.1 specifies the calibration protocol, preprocessing pipeline, codebooks, and baseline configurations. Appendix A.1 gives MPQ hyperparameters, Algorithm 1 and Appendix B.5 provide the algorithm and a Python-style pseudocode reference.

Guidelines:

- The answer [N/A] means that the paper does not include experiments.
- If the paper includes experiments, a [No] answer to this question will not be perceived well by the reviewers: Making the paper reproducible is important, regardless of whether the code and data are provided or not.
- If the contribution is a dataset and/or model, the authors should describe the steps taken to make their results reproducible or verifiable.
- Depending on the contribution, reproducibility can be accomplished in various ways. For example, if the contribution is a novel architecture, describing the architecture fully might suffice, or if the contribution is a specific model and empirical evaluation, it may be necessary to either make it possible for others to replicate the model with the same dataset, or provide access to the model. In general. releasing code and data is often one good way to accomplish this, but reproducibility can also be provided via detailed instructions for how to replicate the results, access to a hosted model (e.g., in the case of a large language model), releasing of a model checkpoint, or other means that are appropriate to the research performed.
- While NeurIPS does not require releasing code, the conference does require all submissions to provide some reasonable avenue for reproducibility, which may depend on the nature of the contribution. For example
  (a) If the contribution is primarily a new algorithm, the paper should make it clear how to reproduce that algorithm.
  (b) If the contribution is primarily a new model architecture, the paper should describe the architecture clearly and fully.
  (c) If the contribution is a new model (e.g., a large language model), then there should either be a way to access this model for reproducing the results or a way to reproduce the model (e.g., with an open-source dataset or instructions for how to construct the dataset).

(d) We recognize that reproducibility may be tricky in some cases, in which case authors are welcome to describe the particular way they provide for reproducibility. In the case of closed-source models, it may be that access to the model is limited in some way (e.g., to registered users), but it should be possible for other researchers to have some path to reproducing or verifying the results.

5. **Open access to data and code**

   Question: Does the paper provide open access to the data and code, with sufficient instructions to faithfully reproduce the main experimental results, as described in supplemental material?

   Answer: [No]

   Justification: We do not release code at submission. The reference Python-style pseudocode in Appendix B.5, the exact update formulas in Appendix B, and the hyperparameters in Appendix A.1 together specify the algorithm sufficiently for reproduction. We commit to releasing the codebase at the camera-ready stage.

   Guidelines:

   - The answer [N/A] means that paper does not include experiments requiring code.
   - Please see the NeurIPS code and data submission guidelines (https://neurips.cc/public/guides/CodeSubmissionPolicy) for more details.
   - While we encourage the release of code and data, we understand that this might not be possible, so [No] is an acceptable answer. Papers cannot be rejected simply for not including code, unless this is central to the contribution (e.g., for a new open-source benchmark).
   - The instructions should contain the exact command and environment needed to run to reproduce the results. See the NeurIPS code and data submission guidelines (https://neurips.cc/public/guides/CodeSubmissionPolicy) for more details.
   - The authors should provide instructions on data access and preparation, including how to access the raw data, preprocessed data, intermediate data, and generated data, etc.
   - The authors should provide scripts to reproduce all experimental results for the new proposed method and baselines. If only a subset of experiments are reproducible, they should state which ones are omitted from the script and why.
   - At submission time, to preserve anonymity, the authors should release anonymized versions (if applicable).
   - Providing as much information as possible in supplemental material (appended to the paper) is recommended, but including URLs to data and code is permitted.

6. **Experimental setting/details**

   Question: Does the paper specify all the training and test details (e.g., data splits, hyperparameters, how they were chosen, type of optimizer) necessary to understand the results?

   Answer: [Yes]

   Justification: Section 6.1 specifies calibration data, preprocessing, codebooks, and baseline variants. The pilot sweep that motivates the hyperparameter values are reported in Figure 5 of Appendix A.1.

   Guidelines:

   - The answer [N/A] means that the paper does not include experiments.
   - The experimental setting should be presented in the core of the paper to a level of detail that is necessary to appreciate the results and make sense of them.
   - The full details can be provided either with the code, in appendix, or as supplemental material.

7. **Experiment statistical significance**

   Question: Does the paper report error bars suitably and correctly defined or other appropriate information about the statistical significance of the experiments?

   Answer: [No]

Justification: Appendix A.7 reports mean and sample standard deviation over three paired C4 calibration draws for 3-bit and 4-bit scalar quantization on four Llama models. Error bars measure calibration-sampling variability. Other experiments report individual runs.

Guidelines:

- The answer [N/A] means that the paper does not include experiments.
- The authors should answer [Yes] if the results are accompanied by error bars, confidence intervals, or statistical significance tests, at least for the experiments that support the main claims of the paper.
- The factors of variability that the error bars are capturing should be clearly stated (for example, train/test split, initialization, random drawing of some parameter, or overall run with given experimental conditions).
- The method for calculating the error bars should be explained (closed form formula, call to a library function, bootstrap, etc.)
- The assumptions made should be given (e.g., Normally distributed errors).
- It should be clear whether the error bar is the standard deviation or the standard error of the mean.
- It is OK to report 1-sigma error bars, but one should state it. The authors should preferably report a 2-sigma error bar than state that they have a 96% CI, if the hypothesis of Normality of errors is not verified.
- For asymmetric distributions, the authors should be careful not to show in tables or figures symmetric error bars that would yield results that are out of range (e.g., negative error rates).
- If error bars are reported in tables or plots, the authors should explain in the text how they were calculated and reference the corresponding figures or tables in the text.

8. **Experiments compute resources**

   Question: For each experiment, does the paper provide sufficient information on the computer resources (type of compute workers, memory, time of execution) needed to reproduce the experiments?

   Answer: [No]

   Justification: Section 6 specifies the timing and memory protocol on a desktop with single RTX 5090 and 64 GB RAM. The Llama-2-70B runs use H200 or L40S cluster but did not record the compute time.

   Guidelines:

   - The answer [N/A] means that the paper does not include experiments.
   - The paper should indicate the type of compute workers CPU or GPU, internal cluster, or cloud provider, including relevant memory and storage.
   - The paper should provide the amount of compute required for each of the individual experimental runs as well as estimate the total compute.
   - The paper should disclose whether the full research project required more compute than the experiments reported in the paper (e.g., preliminary or failed experiments that didn't make it into the paper).

9. **Code of ethics**

   Question: Does the research conducted in the paper conform, in every respect, with the NeurIPS Code of Ethics `https://neurips.cc/public/EthicsGuidelines`?

   Answer: [Yes]

   Justification: The research conforms to the NeurIPS Code of Ethics. The paper studies algorithmic improvements to weight quantization for already-publicly-released language models and does not involve human subjects, sensitive data, or new model releases.

   Guidelines:

   - The answer [N/A] means that the authors have not reviewed the NeurIPS Code of Ethics.
   - If the authors answer [No], they should explain the special circumstances that require a deviation from the Code of Ethics.

- The authors should make sure to preserve anonymity (e.g., if there is a special consideration due to laws or regulations in their jurisdiction).

10. **Broader impacts**

    Question: Does the paper discuss both potential positive societal impacts and negative societal impacts of the work performed?

    Answer: [Yes]

    Justification: Appendix D discusses both potential positive impacts (lower deployment cost, broader accessibility, reduced inference energy footprint) and potential negative impacts (acceleration of harmful language model applications) of the work.

    Guidelines:

    - The answer [N/A] means that there is no societal impact of the work performed.
    - If the authors answer [N/A] or [No], they should explain why their work has no societal impact or why the paper does not address societal impact.
    - Examples of negative societal impacts include potential malicious or unintended uses (e.g., disinformation, generating fake profiles, surveillance), fairness considerations (e.g., deployment of technologies that could make decisions that unfairly impact specific groups), privacy considerations, and security considerations.
    - The conference expects that many papers will be foundational research and not tied to particular applications, let alone deployments. However, if there is a direct path to any negative applications, the authors should point it out. For example, it is legitimate to point out that an improvement in the quality of generative models could be used to generate Deepfakes for disinformation. On the other hand, it is not needed to point out that a generic algorithm for optimizing neural networks could enable people to train models that generate Deepfakes faster.
    - The authors should consider possible harms that could arise when the technology is being used as intended and functioning correctly, harms that could arise when the technology is being used as intended but gives incorrect results, and harms following from (intentional or unintentional) misuse of the technology.
    - If there are negative societal impacts, the authors could also discuss possible mitigation strategies (e.g., gated release of models, providing defenses in addition to attacks, mechanisms for monitoring misuse, mechanisms to monitor how a system learns from feedback over time, improving the efficiency and accessibility of ML).

11. **Safeguards**

    Question: Does the paper describe safeguards that have been put in place for responsible release of data or models that have a high risk for misuse (e.g., pre-trained language models, image generators, or scraped datasets)?

    Answer: [N/A]

    Justification: The paper does not release new pre-trained models, datasets, or other assets with high misuse risk. The contribution is an algorithm for quantizing existing publicly released models.

    Guidelines:

    - The answer [N/A] means that the paper poses no such risks.
    - Released models that have a high risk for misuse or dual-use should be released with necessary safeguards to allow for controlled use of the model, for example by requiring that users adhere to usage guidelines or restrictions to access the model or implementing safety filters.
    - Datasets that have been scraped from the Internet could pose safety risks. The authors should describe how they avoided releasing unsafe images.
    - We recognize that providing effective safeguards is challenging, and many papers do not require this, but we encourage authors to take this into account and make a best faith effort.

12. **Licenses for existing assets**

Question: Are the creators or original owners of assets (e.g., code, data, models), used in the paper, properly credited and are the license and terms of use explicitly mentioned and properly respected?

Answer: [Yes]

Justification: All datasets, models, and tooling used in our experiments are publicly available, cited in the references, and used in accordance with their respective license terms. See Appendix F.

Guidelines:

- The answer [N/A] means that the paper does not use existing assets.
- The authors should cite the original paper that produced the code package or dataset.
- The authors should state which version of the asset is used and, if possible, include a URL.
- The name of the license (e.g., CC-BY 4.0) should be included for each asset.
- For scraped data from a particular source (e.g., website), the copyright and terms of service of that source should be provided.
- If assets are released, the license, copyright information, and terms of use in the package should be provided. For popular datasets, `paperswithcode.com/datasets` has curated licenses for some datasets. Their licensing guide can help determine the license of a dataset.
- For existing datasets that are re-packaged, both the original license and the license of the derived asset (if it has changed) should be provided.
- If this information is not available online, the authors are encouraged to reach out to the asset's creators.

13. **New assets**

Question: Are new assets introduced in the paper well documented and is the documentation provided alongside the assets?

Answer: [N/A]

Justification: This submission does not release new assets. The paper introduces an algorithm (MPQ); it does not release new datasets, pre-trained models, or other artifacts. We commit to releasing an open-source reference implementation of MPQ along with all baseline solver implementations, calibration scripts, and evaluation scripts at the camera-ready stage.

Guidelines:

- The answer [N/A] means that the paper does not release new assets.
- Researchers should communicate the details of the dataset/code/model as part of their submissions via structured templates. This includes details about training, license, limitations, etc.
- The paper should discuss whether and how consent was obtained from people whose asset is used.
- At submission time, remember to anonymize your assets (if applicable). You can either create an anonymized URL or include an anonymized zip file.

14. **Crowdsourcing and research with human subjects**

Question: For crowdsourcing experiments and research with human subjects, does the paper include the full text of instructions given to participants and screenshots, if applicable, as well as details about compensation (if any)?

Answer: [N/A]

Justification: The paper does not involve crowdsourcing or research with human subjects.

Guidelines:

- The answer [N/A] means that the paper does not involve crowdsourcing nor research with human subjects.
- Including this information in the supplemental material is fine, but if the main contribution of the paper involves human subjects, then as much detail as possible should be included in the main paper.

- According to the NeurIPS Code of Ethics, workers involved in data collection, curation, or other labor should be paid at least the minimum wage in the country of the data collector.

15. **Institutional review board (IRB) approvals or equivalent for research with human subjects**

    Question: Does the paper describe potential risks incurred by study participants, whether such risks were disclosed to the subjects, and whether Institutional Review Board (IRB) approvals (or an equivalent approval/review based on the requirements of your country or institution) were obtained?

    Answer: [N/A]

    Justification: The paper does not involve research with human subjects, so IRB approval is not applicable.

    Guidelines:

    - The answer [N/A] means that the paper does not involve crowdsourcing nor research with human subjects.
    - Depending on the country in which research is conducted, IRB approval (or equivalent) may be required for any human subjects research. If you obtained IRB approval, you should clearly state this in the paper.
    - We recognize that the procedures for this may vary significantly between institutions and locations, and we expect authors to adhere to the NeurIPS Code of Ethics and the guidelines for their institution.
    - For initial submissions, do not include any information that would break anonymity (if applicable), such as the institution conducting the review.

16. **Declaration of LLM usage**

    Question: Does the paper describe the usage of LLMs if it is an important, original, or non-standard component of the core methods in this research? Note that if the LLM is used only for writing, editing, or formatting purposes and does *not* impact the core methodology, scientific rigor, or originality of the research, declaration is not required.

    Answer: [N/A]

    Justification: LLMs were used for writing polishing only in this paper.

    Guidelines:

    - The answer [N/A] means that the core method development in this research does not involve LLMs as any important, original, or non-standard components.
    - Please refer to our LLM policy in the NeurIPS handbook for what should or should not be described.