MULTI-BITWIDTH QUANTIZATION FOR LLMS USING ADDITIVE CODEBOOKS

Anonymous authors
Paper under double-blind review

ABSTRACT

As large language models (LLMs) are increasingly deployed across heterogeneous hardware with varying resource constraints, the ability to adaptively manage the performance-efficiency trade-off without retraining is critical. We propose **Drop-by-Drop**, a novel *multi-bitwidth* post-training quantization framework enabling inference-time precision control over LLM weights from a single quantized model. As theoretical motivation, we prove that Gaussian sources are successively refinable under a weighted mean squared error distortion motivated by LLM loss functions: successive refinement across distortion levels incurs no rate penalty relative to encoding at each level separately. Drop-by-Drop approximates this hierarchical structure in practice by incorporating Matryoshka-style supervision into additive codebook training, inducing an ordering in which codebook prefixes yield accurate *partial* reconstructions at each precision level. Furthermore, a block-Hadamard rotation brings the weight distribution closer to our Gaussian source assumption. The result is a single model that serves multiple bitwidths by dropping codebooks, reducing storage and quantization cost relative to the static per-bitwidth models. Across Qwen, LLaMA, Gemma, and Mistral, Drop-by-Drop achieves lower perplexity than state-of-the-art multi-bitwidth methods with competitive zero-shot accuracy, while our specialized kernels further reduce decoding latency.

1 INTRODUCTION

The rapid development of large language models (LLMs) has advanced natural language processing. Advancements in both algorithmic design and large-scale computational infrastructure have enabled open-access models such as LLaMA (Grattafiori et al., 2024), Mistral (Jiang et al., 2023), Gemma (Team et al., 2024), and Qwen (Yang et al., 2025) to achieve strong performance across a wide range of language tasks (Naveed et al., 2025; Ho et al., 2024). Despite their impressive capabilities, LLMs must ultimately be deployed for inference in real-world systems.

Modern LLMs often contain tens or hundreds of billions of parameters, imposing substantial memory and computational demand. Deployment is particularly challenging across heterogeneous platforms that include large cloud servers, personal devices, and edge hardware with strict memory, latency, or power constraints (Yan & Ding, 2025; Zheng et al., 2025). Quantization addresses part of this challenge by compressing model parameters into finite-precision representations, replacing high-precision floating point values. However, most quantization methods produce models at a fixed compression level, which is rarely optimal for all deployment settings. Maintaining separately quantized models for different resource constraints is often impractical due to storage and maintenance overhead. Motivated by this, we explore *multi-bitwidth quantization* for LLM inference, where a single model operates at multiple precision levels without retraining or storing a separate checkpoint per bitwidth. As illustrated in Figure 1, supporting K

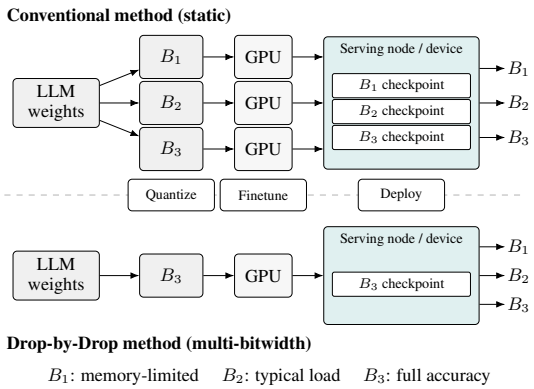


Figure 1: Deploying multiple precision levels

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

bitwidths conventionally requires running quantization and finetuning K distinct times, incurring the overhead of storing K individual checkpoints. In contrast, our approach runs once and stores a single checkpoint that accommodates several operating points, at a storage cost set only by the highest supported bitwidth.

From an information-theoretic perspective, multi-bitwidth quantization naturally relates to the concept of *successive source refinement*, where progressively adding bits yields increasingly accurate reconstructions. While classical results establish successive refinability for certain sources and distortion measures (Equitz & Cover, 1991), its connection to multi-bitwidth quantization for LLMs has not been explored. We bridge this gap by connecting successive refinement theory with practical LLM quantization.

To enable this precision control in LLMs, we propose **Drop-by-Drop**, a novel multi-bitwidth post-training quantization framework that allows inference-time control over the precision of model weights. Our method builds on AQLM (Egiazarian et al., 2024), which represents weights as a sum of codewords drawn from multiple learned codebooks using additive quantization. While AQLM achieves strong accuracy at low bitwidths, it relies on a fixed number of codebooks and therefore produces a static precision profile. Drop-by-Drop addresses this limitation by introducing Matryoshka-style supervision (Kusupati et al., 2022) during codebook training, explicitly encouraging intermediate subsets of codebooks to produce accurate *partial* reconstructions. This design enables progressive compression by simply dropping codebooks during inference, allowing a single quantized model to operate across multiple resource budgets. Our main contributions are summarized below:

1. **Theoretical Foundation:** In Section 4, we establish a theorem proving that Gaussian sources are successively refinable under a weighted mean squared error distortion measure motivated by the loss functions used in LLMs. This extension motivates the design of our algorithmic framework.
2. **Algorithmic Framework:** In Section 5, we propose **Drop-by-Drop**, which enables inference-time precision control from a single quantized model by progressively dropping additive codebooks.
3. **Empirical Evaluation:** In Section 6, we demonstrate that Drop-by-Drop maintains low perplexity and strong accuracy across the supported bitwidths, with lower decoding latency and graceful performance degradation under resource constraints, without retraining or recalibration.

2 PRELIMINARIES

Rate and Distortion. Let W be a discrete random variable taking values in $\mathcal{W}$, with probability mass function $p(w)$, $w \in \mathcal{W}$. In the lossy setting, the source is encoded at rate R (in bits per source symbol) and reconstructed as $\widehat{\mathsf{W}}$. The fidelity of this reconstruction is quantified by a distortion function $d : \mathcal{W} \times \widehat{\mathcal{W}} \to \mathbb{R}$, and performance is measured by the expected distortion $\mathbb{E}[d(\mathsf{W}, \widehat{\mathsf{W}})]$. Let $D \geq 0$ denote the maximum allowable expected distortion. For the distortion function $d(w, \widehat{w})$ defined on $\mathcal{W} \times \widehat{\mathcal{W}}$, the rate-distortion function $R(D)$ and its inverse $D(R)$ are given by

$$R(D) = \min_{p(\widehat{w}|w)} I(\mathsf{W}; \widehat{\mathsf{W}}) \quad \text{and} \quad D(R) = \min_{p(\widehat{w}|w):\, I(\mathsf{W};\widehat{\mathsf{W}}) \leq R} \mathbb{E}\Big[d(\mathsf{W}, \widehat{\mathsf{W}})\Big].$$

The minimization is over all $p(\widehat{w} \mid w)$ subject to $\sum_{w,\widehat{w}} p(w)\, p(\widehat{w} \mid w)\, d(w, \widehat{w}) \leq D$. These are single-letter characterizations; the operational definition is in Cover & Thomas (2006, Ch. 10).

Successive Refinement. A source (W, d) is successively refinable if, for any finite distortion chain $D_1 > D_2 > \cdots > D_K > 0$, there exists a single embedded code whose k-prefix achieves the optimal rate–distortion pair $(R(D_k), D_k)$ for all $k = 1, \ldots, K$, with no rate loss compared to encoding directly at D_k. That is, one code can operate optimally at multiple distortion levels. We define successive refinability for general sources in Definition 1, and for Gaussian sources under mean squared error (MSE) distortion in Theorem 2.

Loss Formulation for LLM Weight Quantization. LLMs are composed of stacked layers, each performing a linear transformation followed by a non-linearity. In this work, we work on quantizing the LLM weights of a pre-trained model. For a given layer, let $\mathbf{W} \in \mathbb{R}^{d_{\text{out}} \times d_{\text{in}}}$ denote its weight matrix, where d_{in} and d_{out} denote the input and output dimensions of the layer, respectively. We have a fixed data matrix $\boldsymbol{X} \in \mathbb{R}^{d_{\text{in}} \times n}$, collected from representative inputs. For the first layer,

$\boldsymbol{X}$ corresponds to token embeddings of the dataset. Whereas, for subsequent layers, $\boldsymbol{X}$ consists of intermediate activations produced by preceding layers. The corresponding layer outputs are $\mathbf{Y} = \mathbf{W}\boldsymbol{X} \in \mathbb{R}^{d_{\text{out}} \times n}$. Notation description is in Appendix A.

Let $\widehat{\mathbf{W}}$ denote a *quantized reconstruction* of $\mathbf{W}$. To evaluate the impact of quantization on the layer outputs, we define the *weighted mean squared error (WMSE)*:

$$d_{\text{WMSE}}(\mathbf{W}, \widehat{\mathbf{W}}) \;\coloneqq\; \mathbb{E}\Big[\big\|(\mathbf{W} - \widehat{\mathbf{W}})\boldsymbol{X}\big\|_F^2\Big].$$

When $\boldsymbol{X} = \boldsymbol{I}$, the identity matrix, this reduces to the standard MSE between $\boldsymbol{W}$ and $\widehat{\boldsymbol{W}}$. However, in practice, we do not have access to the distribution of $\boldsymbol{W}$ and observe only a single realization:

$$L_{\text{WMSE}}(\mathbf{W}, \widehat{\mathbf{W}}) \;\coloneqq\; \frac{1}{d_{\text{out}} n}\big\|(\mathbf{W} - \widehat{\mathbf{W}})\boldsymbol{X}\big\|_F^2 = \frac{1}{d_{\text{out}}}\,\text{tr}\left[(\mathbf{W} - \widehat{\mathbf{W}})\frac{1}{n}\boldsymbol{X}\boldsymbol{X}^\top(\mathbf{W} - \widehat{\mathbf{W}})^\top\right]. \quad (1)$$

3 Background & Related Work

3.1 The Multi-Bitwidth Quantization Landscape

Post-training quantization (PTQ) compresses a pre-trained large language model (LLM) using a small calibration set, avoiding the compute overhead of quantization-aware training (QAT) at billion-parameter scale. Well established PTQ methods compress weights to lower precision with minimal accuracy loss. However, each of these methods fixes a single target bitwidth: serving K operating precisions requires calibrating, storing, and deploying K independent models. Our interest is in the multi-bitwidth setting. Among the vast literature on PTQ, a smaller set of methods achieves multi-bitwidth operation. Closest to our setting are the following: Any-Precision LLM (Park et al., 2024) designs a non-uniform scalar quantizer where lower-bit representations act as nested prefixes of higher-bit ones. AnyBCQ (Park et al., 2026) shares frozen binary bit-planes across precisions. MatGPTQ (Kleinegger et al., 2026) formulates a one-shot PTQ objective across multiple bitwidths, using cross-bit error compensation to generate a sliceable model in a single pass.

AnyBCQ and Any-Precision LLM do not work with the weighted loss in Equation 1, instead they use different adaptations for their own quantizers. AnyBCQ fits its bit-planes via a data-unaware objective and applies a data-aware calibration loss to scale factors in a secondary stage. Meanwhile, Any-Precision LLM inherits SqueezeLLM's diagonal approximation of the empirical Fisher information matrix (Kim et al., 2023), which omits the off-diagonal terms that couple the weights (Kim et al., 2025). Another weakness is that all three frameworks rely on scalar quantization (SQ), whose performance is tightly constrained by information-theoretic bounds (van der Ouderaa et al., 2026). In the low-bit regime, vector quantization (VQ) achieves a lower distortion than SQ at the same rate (Gersho & Gray, 2012, Ch. 10). Empirically, scaling the vector dimension, such as grouping weights into 2D, 4D, and 8D blocks, yields monotonic improvements in signal-to-quantization-noise ratio (Van Baalen et al., 2024; Wang et al., 2026b), resulting in lower perplexity (Egiazarian et al., 2024; Tseng et al., 2024; Zhou et al., 2025) and higher zero-shot accuracy (Liu et al., 2024; Wang et al., 2026a). Therefore, we utilize AQLM as our additive-codebook building block, quantizing weights in 8-dimensional vectors and building on their WMSE objective in Equation 1.

3.2 Additive Quantization and AQLM

Among vector approaches, additive quantization (AQ) (Babenko & Lempitsky, 2014) removes the strict ordering imposed by residuals by representing weights as *sums* of multiple independently selected vectors drawn from learned codebooks. These components are optimized jointly rather than sequentially, resulting in a more flexible representation with higher expressive capacity at extreme compression ratios. Originally developed for approximate nearest neighbor search, AQ improves approximation quality over standard product and scalar quantization by reconstructing each weight group as a sum of vectors from multiple codebooks (Babenko & Lempitsky, 2014). Building on this idea, AQLM applies multi-codebook

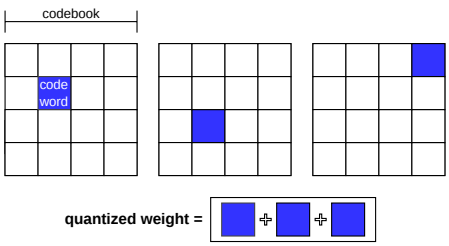


Figure 2: Illustration of additive codebook-based reconstruction.

additive PTQ to compress LLM weights, representing each weight tensor as a sum of quantized vectors selected from distinct codebooks.

Given a linear layer with weight matrix $\mathbf{W} \in \mathbb{R}^{d_{\text{out}} \times d_{\text{in}}}$ and data inputs $\boldsymbol{X} \in \mathbb{R}^{d_{\text{in}} \times n}$, AQLM aims to find quantized weights $\widehat{\mathbf{W}}$ that minimize the squared error between the original and quantized outputs: $\min_{\widehat{\mathbf{W}}} \left\| \mathbf{W}\boldsymbol{X} - \widehat{\mathbf{W}}\boldsymbol{X} \right\|_F^2$. In AQLM, the weight matrix $\mathbf{W}$ is split into groups of g consecutive elements. Each group is approximated by the sum of M vectors, each selected from a learned codebook $C_1, C_2, \ldots, C_M$. Each codebook contains 2^B codewords, where B is the number of bits used to index the codewords. The selection for each group is encoded by M one-hot vectors b_{ijm}, where b_{ijm} indicates the codeword chosen from the m-th codebook for the j-th group in the i-th output row. Formally, a group is reconstructed as $\sum_{m=1}^{M} C_m b_{ijm}$. Thus, the assignments b specify the combination of codewords whose sum reconstructs each weight group. Figure 2 shows how each weight group is reconstructed by summing selected codewords from multiple codebooks. To form the full quantized weight matrix $\widehat{\mathbf{W}}$, AQLM concatenates the reconstructed groups along the input dimension, where $\oplus$ denotes vector concatenation across the d_{in}/g groups per output row: $\widehat{\mathbf{W}}_i = \sum_{m=1}^{M} C_m b_{i,1,m} \oplus \cdots \oplus \sum_{m=1}^{M} C_m b_{i,d_{\text{in}}/g,m}$. The optimization process of AQLM is described in Appendix B. Next, we review Matryoshka Representation Learning, the supervision scheme we adapt to AQLM's additive codebooks to achieve the multi-bitwidth inference.

3.3 Matryoshka Representation Learning

Matryoshka Representation Learning (Kusupati et al., 2022) enables a single model to adaptively use representations at varying fidelity by encouraging intermediate prefixes to approximate the final output, supporting coarse-to-fine inference. Related work includes Matryoshka Query Transformer for Large Vision-Language Models (Hu et al., 2024), which applies the same principle in multi-modal settings, and MatQuant (Nair et al., 2025), which extends Matryoshka supervision to quantized models via QAT across multiple bitwidths. MatQuant exploits a hierarchy that is inherent to the integer datatype itself: the two most significant bits of int4 equal int2, and int4 in turn forms the most significant bits of int8, so lower-precision models are recovered by bit-slicing. MatGPTQ (Kleinegger et al., 2026) replaces MatQuant's QAT with the one-shot PTQ objective described above, casting the same bit-slicing hierarchy as a post-training rather than a training-time procedure. We extend Matryoshka supervision to additive vector codebooks. Since additive codebooks are optimized jointly and have no inherent ordering, we consider each codebook prefix to serve as a standalone lower-bitwidth model, rather than dropping codebooks post-hoc.

4 Our Theoretical Contribution

We now characterize the *successive refinability of Gaussian sources under weighted MSE*, culminating in Theorem 1. We model each weight matrix $\mathbf{W}$ as Gaussian entries. This assumption is standard in rate-distortion analysis (Sakrison, 1968), admitting exact rate-distortion characterizations and known successive refinability under MSE. In practice, LLM weight distributions are only *approximately* Gaussian: as shown in Appendix C, raw weights follow a broadly bell-shaped profile but with heavier tails. Section 5 discusses how a Block-Hadamard rotation reduces this gap prior to quantization. At the same time, practical PTQ methods optimize data-aware weighted MSE objectives (WMSE), such as Equation 1, where the weighting arises from the data matrix $\boldsymbol{X}$. This provides theoretical motivation for designing progressive and adaptive quantization schemes. Full proofs and visuals are in Appendix E.

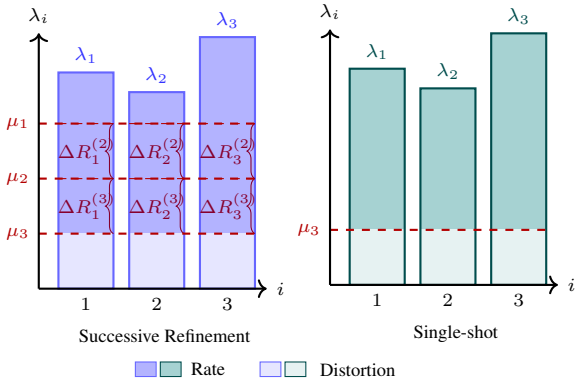


Figure 3: Rate-distortion equivalence

Theorem 1. *Let* W *be a zero-mean Gaussian random matrix with i.i.d. (independent and identically distributed) entries, and consider the weighted mean squared error distortion with a fixed matrix* $\boldsymbol{X}$*,* $d_{\mathrm{WMSE}}(\mathsf{W}, \widehat{\mathsf{W}}) = \mathbb{E}\Big[\|(\mathsf{W} - \widehat{\mathsf{W}})\boldsymbol{X}\|_F^2\Big]$*. Then, the source* $(\mathsf{W}, d_{\mathrm{WMSE}})$ *is successively refinable.*

The central challenge is that the WMSE distortion couples all entries of W through the fixed data matrix $\boldsymbol{X}$, making it non-obvious that successive refinement is achievable without a rate penalty. The proof resolves this by reducing to a standard Gaussian MSE problem, then revealing the nested structure of reverse water-filling. The reverse water-filling method is an optimization technique used in information theory and communication systems which allocates more bits (rate) to *high-variance* channels to minimize total distortion under a rate constraint.

Proof Sketch. Introduce the transformed source $\mathsf{Y} = \mathsf{W}\boldsymbol{X}$. The rate-distortion function for W under d_{WMSE} equals the standard Gaussian MSE rate-distortion function for Y, with a one-to-one correspondence between their optimal codebooks at every distortion level D. This follows from the distortion equivalence: the optimal $\widehat{\mathsf{W}}$ is recovered by inverting $\mathsf{Y} = \mathsf{W}\boldsymbol{X}$, and satisfies $d_{\mathrm{WMSE}}(\mathsf{W}, \widehat{\mathsf{W}}) = \mathbb{E}\Big[\big\|(\mathsf{W} - \widehat{\mathsf{W}})\boldsymbol{X}\big\|_F^2\Big] = \mathbb{E}\Big[\big\|\mathsf{Y} - \widehat{\mathsf{Y}}\big\|_F^2\Big]$, by Theorem 4.

It therefore suffices to solve the standard MSE rate-distortion problem for Y. To do so, we vectorize $\mathsf{y} \coloneqq \mathrm{vec}(\mathsf{Y})$ and compute that y is a zero-mean coloured Gaussian with covariance $\boldsymbol{\Sigma}_{\mathsf{y}} = \big(\sigma_w^2 \boldsymbol{X}^\top \boldsymbol{X}\big) \otimes \boldsymbol{I}_{d_{\mathrm{out}}}$, where $\otimes$ denotes the Kronecker product. Since y is correlated, reverse water-filling cannot be applied directly. Diagonalizing via eigen-decomposition $\boldsymbol{\Sigma}_{\mathsf{y}} = \boldsymbol{U}\,\mathrm{diag}(\lambda_1, \lambda_2, \ldots)\,\boldsymbol{U}^\top$ yields independent scalar Gaussian components $\mathsf{v}_i \sim \mathcal{N}(0, \lambda_i)$, to which reverse water-filling applies. The optimal rate-distortion tradeoff for y under MSE is therefore achieved by assigning per-component distortion $D_i = \min(\mu, \lambda_i)$, where the "water level" μ is chosen so that $\sum_i D_i = D$ (Theorem 4). The water level μ vertically partitions the variance λ_i of each component into two distinct segments: the portion below the water level represents the remaining distortion, while the portion extending above the water level determines the allocated rate.

To establish successive refinability, we use superscripts k to index refinement stages and subscripts i to index scalar components. For nested overall distortion levels $D^{(1)} > \cdots > D^{(K)}$, reverse water-filling assigns per-component distortions $D_i^{(k)} = \min(\mu_k, \lambda_i)$, corresponding to a strictly decreasing sequence of water levels $\mu_1 > \cdots > \mu_K$. For each component v_i, define mutually independent Gaussian increments

$$\mathsf{z}_i^{(k)} \sim \mathcal{N}\Big(0,\, D_i^{(k-1)} - D_i^{(k)}\Big), \qquad D_i^{(0)} \coloneqq \lambda_i,$$

and construct successive reconstructions via $\widehat{\mathsf{v}}_i^{(0)} = 0$, $\widehat{\mathsf{v}}_i^{(k)} = \widehat{\mathsf{v}}_i^{(k-1)} + \mathsf{z}_i^{(k)}$. Each increment $\mathsf{z}_i^{(k)}$ carries exactly the new information needed to improve the reconstruction of v_i from distortion $D_i^{(k-1)}$ to the finer distortion $D_i^{(k)}$; it is independent of all previous increments and of the current reconstruction $\widehat{\mathsf{v}}_i^{(k-1)}$. The incremental rate $\Delta R_i^{(k)}$ is therefore the number of bits required to describe $\mathsf{z}_i^{(k)}$, i.e. the mutual information between v_i and the refined reconstruction $\widehat{\mathsf{v}}_i^{(k)}$ given the previous one: $\Delta R_i^{(k)} = I\Big(\mathsf{v}_i; \widehat{\mathsf{v}}_i^{(k)} \,\Big|\, \widehat{\mathsf{v}}_i^{(k-1)}\Big) = \frac{1}{2}\log\frac{D_i^{(k-1)}}{D_i^{(k)}}$. By construction, $\mathbb{E}[(\mathsf{v}_i - \widehat{\mathsf{v}}_i^{(k)})^2] = D_i^{(k)}$ at every stage, and since each increment is independent of all previous ones, the Markov chain $\widehat{\mathsf{v}}_i^{(1)} \to \cdots \to \widehat{\mathsf{v}}_i^{(K)} \to \mathsf{v}_i$ holds (Theorem 5). The below sum then confirms there is no rate penalty:

$$\sum_{k=1}^{K} \Delta R_i^{(k)} = \frac{1}{2}\log\frac{\lambda_i}{D_i^{(1)}} + \frac{1}{2}\log\frac{D_i^{(1)}}{D_i^{(2)}} + \cdots + \frac{1}{2}\log\frac{D_i^{(K-1)}}{D_i^{(K)}} = \frac{1}{2}\log\frac{\lambda_i}{D_i^{(K)}},$$

which equals exactly the single-shot rate at $D_i^{(K)}$. As illustrated in Figure 3, the $\Delta R^{(k)}$ brackets partition the total rate between refinement stages identically to single-shot coding at D_K, confirming that refinement incurs no rate overhead. Finally, at each stage k, the reconstruction $\widehat{\mathsf{W}}^{(k)}$ is recovered from $\widehat{\mathsf{Y}}^{(k)}$ by inverting $\mathsf{Y} = \mathsf{W}\boldsymbol{X}$ (depending on the shape of $\boldsymbol{X}$, see Theorem 4). Since this inversion map is identical at every stage k, the Markov chain on y carries over to W, $\widehat{\mathsf{W}}^{(1)} \to \cdots \to \widehat{\mathsf{W}}^{(K)} \to \mathsf{W}$, and the distortion equivalence gives $d_{\mathrm{WMSE}}(\mathsf{W}, \widehat{\mathsf{W}}^{(k)}) = D_k$ at each stage, establishing successive refinability of $(\mathsf{W}, d_{\mathrm{WMSE}})$. □

5 Our Application: Drop-by-Drop (DbyD)

Motivated by our information-theoretic result, we now focus on a *practical* quantization scheme that supports multiple rate–distortion operating points without retraining, under the same distortion measure $(\mathbf{W}, d_{\mathrm{WMSE}})$ used during quantization. AQLM naturally instantiates this structure: each additive codebook can correspond to a refinement layer. However, it is our Drop-by-Drop loss that ensures this *droppable* property holds in practice, where codebooks can be dropped at inference to provide a smooth tradeoff between compression rate and reconstruction quality.

We introduce the **Drop-by-Drop loss**, a training objective inspired by Matryoshka Representation Learning that modifies the codebook update phase of the AQLM pipeline. It enforces a hierarchical organization in which early codebooks capture coarse but fundamental weight features and later ones conduct progressively finer refinements. To induce this ordering, we supervise *partial reconstructions*: for each stage $k = 1, \ldots, M$, we reconstruct the weights from only the first k codebooks, so that C_1 contributes at every level and the representation refines incrementally. Figure 4 depicts these stages as nested boxes, where nesting encodes prefix containment and darker shading indicates higher distortion; sizes and shades are illustrative. Here, $C_m \in \mathbb{R}^{g \times 2^B}$ is the m-th codebook and $b_{ijm} \in \{0,1\}^{2^B}$ is a one-hot vector indicating the selected codeword:

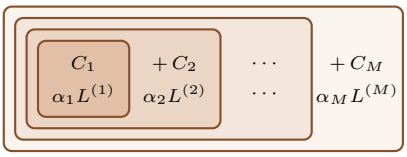


Figure 4: Visualization of $\mathcal{L}_{\mathrm{DbyD}}$.

$$\widehat{\mathbf{W}}_i^{(k)} = \sum_{m=1}^{k} C_m b_{i,1,m} \oplus \cdots \oplus \sum_{m=1}^{k} C_m b_{i,d_{\mathrm{in}}/g,m}$$

This yields an intermediate approximation $\widehat{\mathbf{W}}^{(k)}$, exposing the model during training to all deployment-time precision levels. For each partial reconstruction, we measure distortion using a task-weighted MSE:

$$L^{(k)} = \frac{1}{d_{\mathrm{out}}\, n} \left\| (\mathbf{W} - \widehat{\mathbf{W}}^{(k)})\, \boldsymbol{X} \right\|_F^2 = \frac{1}{d_{\mathrm{out}}} \operatorname{tr}\left[(\mathbf{W} - \widehat{\mathbf{W}}^{(k)}) \frac{1}{n} \boldsymbol{X}\boldsymbol{X}^\top (\mathbf{W} - \widehat{\mathbf{W}}^{(k)})^\top \right].$$

The overall Drop-by-Drop objective aggregates these distortions across refinement stages: $\mathcal{L}_{\mathrm{DbyD}} = \sum_{k=1}^{M} \alpha_k\, L^{(k)}$, where $\alpha_k \geq 0$ are weighting coefficients, normalized such that $\sum_{k=1}^{M} \alpha_k = 1$, controlling the relative importance of coarse versus fine reconstructions. This training logic is summarized in Algorithm 1, and the full algorithm can be found in Appendix F where coloured text shows the changes made to AQLM.

Furthermore, we include a block-wise Walsh-Hadamard rotation as an optional pre-processing step. It reduces the excess kurtosis of the weight distribution, bringing it closer to the Gaussian source assumed by Theorem 1. The analysis is in Appendix C.

At inference time, Drop-by-Drop lets a single deployed model be reconstructed at multiple precisions without retraining. Given a target budget, we dequantize the weights from a prefix of the first k codebooks ($k \leq M$), where k is specified when the model is loaded.

6 Experiments

Experimental Setup & Hyperparameters. Our experiments were conducted on multi-GPU systems equipped with NVIDIA L40S GPUs. We mostly used 4 GPUs with CUDA support. We perform quantization using $M \in \{3, 4, 5\}$ codebooks, each with a bitwidth of 8 and a group size of 8. We used 1024 calibration samples with a validation split of 32 samples, optimizing a relative MSE objective with a tolerance of 0.01. All quantized models are evaluated against the original FP16 base model using a batch size of 8.

Method Definitions. We follow AQLM and partition the weights into groups of 8, applying additive quantization with K codebooks of 8 bits each per group. Selecting K codebooks per group therefore yields an effective rate of K bits per weight. Hence, the 3×8, 4×8, and 5×8 settings, where the first number denotes the number of codebooks and the second the bit-width of each, correspond to 3-, 4-, and 5-bit configurations, respectively. For clarity, we define the compared methods:

Algorithm 1 Drop-by-Drop Quantization for Resource-Constrained Inference

Require: model, data

X_{block} := model.input_embeddings(data)
for layer $\in$ linear_layers(block) **do**
  W := layer.weight, d_{in} := columns(W)
  $XX^\top$:= accumulate_hessian(layer, X_{block})
  Select block size B_H and sample random signs $S \sim \mathcal{U}(\{-1, 1\})^{d_{\text{in}}}$
  $W \leftarrow \text{BlockFWHT}_{\text{1D}}(W \odot S, B_H)$
  $XX^\top \leftarrow \text{BlockFWHT}_{\text{2D}}(XX^\top \odot (SS^\top), B_H)$
  C, b, s := k_means_initialize(W)
  while loss improves by at least τ **do**
    for $k = 1, \ldots, M$ **do**
      $\widehat{W}^{(k)} \leftarrow$ partial reconstruction using the first k codebooks
      $L^{(k)} \leftarrow \frac{1}{d_{\text{out}}} \operatorname{tr}\left[(W - \widehat{W}^{(k)}) \left(\frac{1}{n} XX^\top\right) (W - \widehat{W}^{(k)})^\top\right]$
    end for
    Total loss: $\mathcal{L}_{\text{DbyD}} = \sum_k \alpha_k L^{(k)}$
    Update codebooks C by gradient descent, update assignments b by beam search
  end while
  layer.weight := AQLMFormat(C, b, s)
  layer.rotation_signs := S, layer.block_size := B_H
end for

1. *AQLM (Individual)*: Separate AQLM models trained independently per target configuration. This is the ideal performance bound, but requires training and storing N models for N levels.

2. *AQLM (Drop)*: An AQLM model trained at maximum quantization (e.g., 5×8), with codebooks dropped at inference time to simulate lower quantization levels.

3. *Drop-by-Drop (Uniform)*: A single model trained once at maximum quantization using the proposed Drop-by-Drop method, designed to support codebook reduction at inference time. Uniform Matryoshka loss weights are used, **Uni**: $\alpha_1 = \alpha_2 = \alpha_3 = \alpha_4 = \alpha_5 = \frac{1}{5}$.

4. *Drop-by-Drop (Weighted)*: Variants of DbyD using non-uniform Matryoshka loss weights to better approximate the performance of individually tuned AQLM models. Based on our ablation studies in Appendix G, we report **35W**: $\alpha_3 = \alpha_5 = \frac{1}{2}$, all other $\alpha_k = 0$.

5. *Drop-by-Drop (Hadamard + Weighted)*: The DbyD (35W) variant with a block-wise Walsh-Hadamard rotation applied to the weights prior to quantization.

Evaluation Benchmarks. We evaluate Drop-by-Drop across a diverse set of LLMs spanning multiple architectures and scales, including GEMMA-2B, META-LLAMA-3-8B, MISTRAL-7B-V0.3, and the QWEN2.5 family (0.5B-INSTRUCT, 3B, 7B-INSTRUCT, 14B-INSTRUCT, and 32B-INSTRUCT). Intrinsic language modeling performance is measured using perplexity on C4 and WIKITEXT-2 (Raffel et al., 2020; Merity et al., 2016). Extrinsic evaluation is conducted via zero-shot accuracy on commonsense and reasoning benchmarks, including ARC (Easy and Challenge), HELLASWAG, WINOGRANDE, and PIQA (Clark et al., 2018; Zellers et al., 2019; Sakaguchi et al., 2021; Bisk et al., 2020), reported as the mean across these five tasks. Perplexity trends agree across WIKITEXT-2 and C4, so the main text reports C4 perplexity and mean zero-shot accuracy. WIKITEXT-2 perplexity, per-task accuracy, and the numerical values behind every figure are in Appendix H. All our results are averaged over *three* independent runs.

DbyD vs. AQLM comparison. Figure 5 and Figure 6 compare performance across quantization levels. **AQLM (Ind.)** serves as the specialized performance bound, while **AQLM (Drop)** provides a baseline illustrating the behavior of standard AQLM when codebooks are removed without hierarchical training. As codebooks are dropped, AQLM (Drop) diverges catastrophically from the specialized bound, particularly at the 3×8 level, confirming that standard AQLM does not provide a multi-bitwidth setting. In contrast, the Drop-by-Drop variants consistently outperform this baseline. The **DbyD (35W)** configuration substantially narrows the gap to the individually trained AQLM

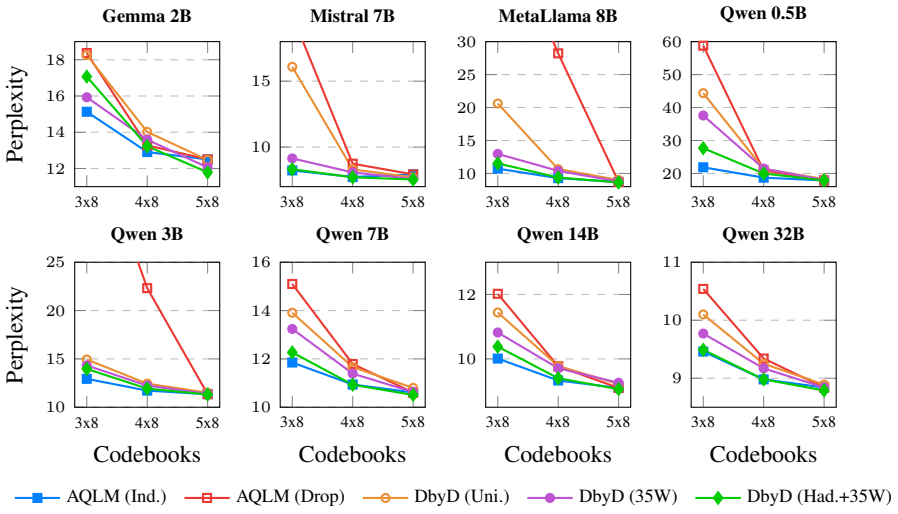


Figure 5: Perplexity on C4 (↓). Extreme outliers for AQLM (Drop) are clipped.

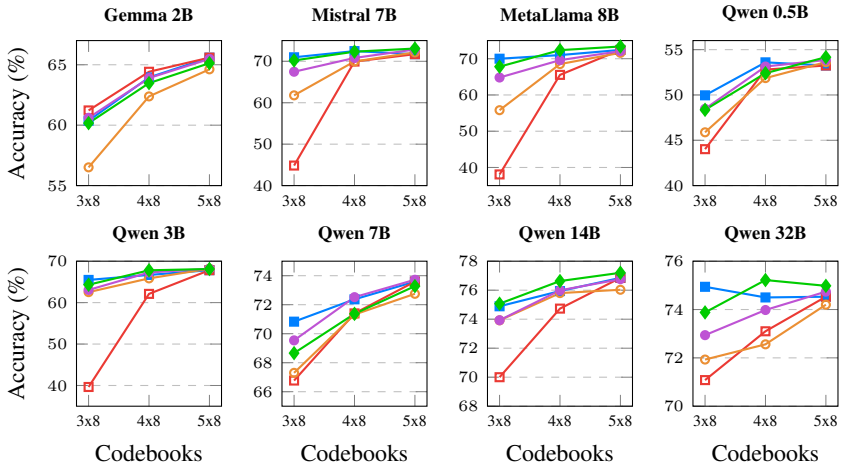


Figure 6: Average zero-shot accuracy (↑) across 5 aforementioned tasks

(Ind.) models, showing that targeted Matryoshka loss weighting lets a single flexible model serve multiple bitwidths with only a modest loss relative to training each bitwidth separately. Applying a block-Hadamard rotation before the WMSE-based supervision, **DbyD (Had.+35W)**, narrows this gap further. In perplexity, DbyD (Had.+35W) is the strongest DbyD variant on nearly all models and bitwidths, approaching AQLM (Ind.) and substantially outperforming AQLM (Drop); the exception is GEMMA-2B, where DbyD (35W) is better at 3×8 on both WIKITEXT2 and C4. Zero-shot accuracy follows a similar trend: at 4×8 and 5×8, DbyD (Had.+35W) closely matches AQLM (Ind.) within standard error.

DbyD vs. SOTA comparison. We compare DbyD (Had.+35W) against three closely related state-of-the-art (SOTA) post-training quantization methods that support multi-bitwidth inference: AnyBCQ, AnyPrecLLM, and MatGPTQ. DbyD offers advantages across several dimensions. *Quality:* DbyD achieves the lowest perplexity at nearly every evaluated precision across all model scales (Table 1, Appendix I.1), while maintaining zero-shot accuracy comparable to the SOTA methods (Appendix I.2). *Storage:* a single DbyD checkpoint supports all evaluated bitwidths, with disk usage comparable to or lower than the corresponding SOTA checkpoints (Figure 7). *Latency:* our `codeNx8` kernels outperform every evaluated SOTA kernel across all layers and bitwidths, with speedups of 1.02–1.46×. The online Hadamard rotation adds negligible overhead (Figure 8; Appendix I.3).

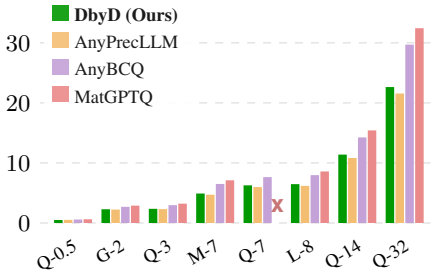


Figure 7: SOTA disk usage (GB) comparison (↓).

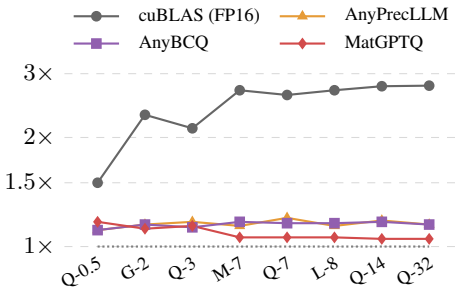


Figure 8: Latency speedup of `codeNx8_Had`.

Quantization resources: DbyD quantizes all evaluated models on L40S GPUs, whereas AnyPrecLLM and MatGPTQ require an H100 for QWEN2.5-32B, and AnyBCQ requires an H100 for both QWEN2.5-14B and QWEN2.5-32B.

Table 1: SOTA perplexity comparison (↓) on C4.

Method	Bits	Qwen2.5					Gemma	Mistral	Llama
		0.5B	3B	7B	14B	32B	2B	7B	8B
DbyD (Had.+35W)	5	**17.87**	**11.32**	**10.51**	**9.06**	**8.79**	**11.79**	**7.54**	**8.63**
AnyBCQ		24.70	14.96	*nan*	11.96	11.63	25.66	17.48	10.24
AnyPrecLLM		22.44	13.77	12.93	11.01	10.70	13.67	8.52	9.68
MatGPTQ		23.25	13.62	–	11.07	10.59	22.62	9.18	9.71
DbyD (Had.+35W)	4	**19.99**	**11.93**	**10.93**	**9.40**	**8.98**	**13.23**	**7.71**	**9.38**
AnyBCQ		25.89	15.36	*nan*	12.15	11.78	25.85	14.28	10.81
AnyPrecLLM		24.15	14.26	13.33	11.28	10.89	14.40	8.67	10.28
MatGPTQ		26.72	14.22	–	11.47	10.81	23.25	9.95	10.60
DbyD (Had.+35W)	3	**27.67**	**13.96**	**12.27**	**10.38**	**9.49**	**17.07**	**8.32**	**11.53**
AnyBCQ		30.62	16.86	*nan*	12.98	12.30	27.79	10.49	13.34
AnyPrecLLM		34.78	16.71	15.52	12.44	11.64	17.58	9.39	13.14
MatGPTQ		80.06	134.12	–	13.23	11.90	52.50	11.95	14.93

nan denotes numerical failure with official code and default settings; – denotes a non-converging configuration.

7 Discussion and Future Work

Discussion. Drop-by-Drop demonstrates that a single quantized model can serve multiple resource budgets without retraining or maintaining separate checkpoints. Conventional methods such as AQLM require a separate model per target configuration, which in our setup means three independently trained and stored models at 3×8, 4×8, and 5×8; DbyD requires a single training run and a single checkpoint. Against state-of-the-art multi-bitwidth methods, it achieves the lowest perplexity at nearly every precision with comparable zero-shot accuracy, a comparable or smaller storage footprint, and lower decoding latency through our `codeNx8` kernels. The precision-performance trade-off is selected at inference time simply by the number of codebooks used, degrading gracefully as the bitwidth decreases.

Limitations and Future Work. While Drop-by-Drop closely approaches individually trained models across the evaluated bitwidths, performance at even lower bitwidths leaves room for further improvement. Our current implementation builds on AQLM's additive codebooks; extending Drop-by-Drop to other quantization schemes, including ones that are cheaper/easier to quantize, is a natural direction. More broadly, the theoretical motivation based on successive refinability for Gaussian sources under weighted mean-squared error could be extended to other source distributions and distortion measures. Finally, the proposed multi-bitwidth representation enables new deployment strategies, including selecting bitwidths based on available memory, compute, or latency budgets, allocating different precisions across layers according to their sensitivity or resource constraints, and dynamically trading model quality for inference speed as deployment conditions change.

AI USE STATEMENT

In this work, we used generative AI tools to *proofread* our theoretical proof. We have **not** used generative AI tools for generating synthetic data sets, helping develop theoretical models or conceptual frameworks, formulating mathematical claims, providing critical ingredients for proving mathematical claims, assisting in the writing of proofs, proposing or refining hypotheses, designing or providing feedback on research methodology or experiments, implementing methods, cleaning and reformating dataset, supporting qualitative and thematic data analysis, or interpreting results; and assistance with translation is not applicable to this work.

We used generative AI tools to identify relevant literature, to optimize software implementation, edit LaTeX, and proofread writing. We have reviewed all AI-assisted work. We checked LLM suggested research papers and conducted our own literature reading. All LLM code and hyperparameter suggestions were tested and verified. We take responsibility for the final content of this work, including text, claims or artifacts produced with the aid of generative AI.

REPRODUCIBILITY STATEMENT

For our theoretical results, Appendices D and E give the assumptions and full proofs behind the main theorem in Section 4. For our practical design, our implementation builds on the AQLM codebase, with modifications released at `https://anonymous.4open.science/r/DropbyDrop2026/`, primarily affecting `aq_engine.py`, `src/aq.py`, `src/modelutils.py` and `quantize.sh`. The experimental setup, hyperparameters, and evaluation benchmarks are listed in Section 6. The repository contains the code needed to reproduce our quantized models; however the checkpoints and specialized inference kernels (Appendix I.3) will be made publicly available on Hugging Face and GitHub upon publication.

REFERENCES

Saleh Ashkboos, Amirkeivan Mohtashami, Maximilian L Croci, Bo Li, Pashmina Cameron, Martin Jaggi, Dan Alistarh, Torsten Hoefler, and James Hensman. Quarot: Outlier-free 4-bit inference in rotated llms. *Advances in Neural Information Processing Systems*, 37:100213–100240, 2024.

Artem Babenko and Victor Lempitsky. Additive quantization for extreme vector compression. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pp. 931–938, 2014.

Yonatan Bisk, Rowan Zellers, Jianfeng Gao, Yejin Choi, et al. Piqa: Reasoning about physical commonsense in natural language. In *Proceedings of the AAAI conference on artificial intelligence*, volume 34, pp. 7432–7439, 2020.

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.

Shuangyi Chen and Ashish J Khisti. Adaquantlm: Llm quantization with adaptive bit-widths. In *Workshop on Machine Learning and Compression, NeurIPS 2024*, 2024.

Yongjian Chen, Tao Guan, and Cheng Wang. Approximate nearest neighbor search by residual vector quantization. *Sensors*, 10(12):11259–11273, 2010.

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 preprint arXiv:1803.05457*, 2018.

Thomas M. Cover and Joy A. Thomas. *Elements of information theory*. John Wiley & Sons, 2 edition, 2006.

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

William Howard Robinson Equitz. *Successive Refinement of Information*. PhD thesis, Stanford University, 1989.

William HR Equitz and Thomas M Cover. Successive refinement of information. *IEEE Transactions on information theory*, 37(2):269–275, 1991.

Allen Gersho and Robert M Gray. *Vector quantization and signal compression*. Springer Science & Business Media, 2012.

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.

Anson Ho, Tamay Besiroglu, Ege Erdil, David Owen, Robi Rahman, Zifan C Guo, David Atkinson, Neil Thompson, and Jaime Sevilla. Algorithmic progress in language models. *Advances in Neural Information Processing Systems*, 37:58245–58283, 2024.

Wenbo Hu, Zi-Yi Dou, Liunian H Li, Amita Kamath, Nanyun Peng, and Kai-Wei Chang. Matryoshka query transformer for large vision-language models. *Advances in Neural Information Processing Systems*, 37:50168–50188, 2024.

Albert Q. 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, 2023. URL `https://arxiv.org/abs/2310.06825`.

Jinuk Kim, Marwa El Halabi, Wonpyo Park, Clemens JS Schaefer, Deokjae Lee, Yeonhong Park, Jae W Lee, and Hyun Oh Song. Guidedquant: Large language model quantization via exploiting end loss guidance. *arXiv preprint arXiv:2505.07004*, 2025.

Sehoon Kim, Coleman Hooper, Amir Gholami, Zhen Dong, Xiuyu Li, Sheng Shen, Michael W Mahoney, and Kurt Keutzer. Squeezellm: Dense-and-sparse quantization. *arXiv preprint arXiv:2306.07629*, 2023.

Maximilian Kleinegger, Elvir Crnčević, and Dan Alistarh. Matgptq: Accurate and efficient post-training matryoshka quantization. *arXiv preprint arXiv:2602.03537*, 2026.

Aditya Kusupati, Gantavya Bhatt, Aniket Rege, Matthew Wallingford, Aditya Sinha, Vivek Ramanujan, William Howard-Snyder, Kaifeng Chen, Sham Kakade, Prateek Jain, et al. Matryoshka representation learning. *Advances in Neural Information Processing Systems*, 35:30233–30249, 2022.

Deokjae Lee and Hyun Oh Song. Q-palette: Fractional-bit quantizers toward optimal bit allocation for efficient llm deployment. *Advances in Neural Information Processing Systems*, 38:11525–11558, 2026.

Yifei Liu, Jicheng Wen, Yang Wang, Shengyu Ye, Li Lyna Zhang, Ting Cao, Cheng Li, and Mao Yang. Vptq: Extreme low-bit vector post-training quantization for large language models. In *Proceedings of the 2024 Conference on Empirical Methods in Natural Language Processing*, pp. 8181–8196, 2024.

Vladimir Malinovskii, Andrei Panferov, Ivan Ilin, Han Guo, Peter Richtárik, and Dan Alistarh. Higgs: Pushing the limits of large language model quantization via the linearity theorem. In *Proceedings of the 2025 Conference of the Nations of the Americas Chapter of the Association for Computational Linguistics: Human Language Technologies (Volume 1: Long Papers)*, pp. 10857–10886, 2025.

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

Pranav Nair, Puranjay Datta, Jeff Dean, Prateek Jain, and Aditya Kusupati. Matryoshka quantization. *arXiv preprint arXiv:2502.06786*, 2025.

Humza Naveed, Asad Ullah Khan, Shi Qiu, Muhammad Saqib, Saeed Anwar, Muhammad Usman, Naveed Akhtar, Nick Barnes, and Ajmal Mian. A comprehensive overview of large language models. *ACM Transactions on Intelligent Systems and Technology*, 16(5):1–72, 2025.

Gunho Park, Jeongin Bae, Beomseok Kwon, Byeongwook Kim, Se Jung Kwon, and Dongsoo Lee. Anybcq: Hardware efficient flexible binary-coded quantization for multi-precision llms. In *International Conference on Learning Representations*, pp. 125106–125122, 2026.

Yeonhong Park, Jake Hyun, SangLyul Cho, Bonggeun Sim, and Jae W Lee. Any-precision llm: Low-cost deployment of multiple, different-sized llms. *arXiv preprint arXiv:2402.10517*, 2024.

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

Keisuke Sakaguchi, Ronan Le Bras, Chandra Bhagavatula, and Yejin Choi. Winogrande: An adversarial winograd schema challenge at scale. *Communications of the ACM*, 64(9):99–106, 2021.

D Sakrison. The rate distortion function of a gaussian process with a weighted square error criterion (corresp.). *IEEE Transactions on Information Theory*, 14(3):506–508, 1968.

Gemma Team, Thomas Mesnard, Cassidy Hardin, Robert Dadashi, Surya Bhupatiraju, Shreya Pathak, Laurent Sifre, Morgane Rivière, Mihir Sanjay Kale, Juliette Love, et al. Gemma: Open models based on gemini research and technology. *arXiv preprint arXiv:2403.08295*, 2024.

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.

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

Tycho FA van der Ouderaa, Mart Van Baalen, Paul Whatmough, and Markus Nagel. Leech lattice vector quantization for efficient llm compression. *arXiv preprint arXiv:2603.11021*, 2026.

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

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

Xiao Yan and Yi Ding. Are we there yet? a measurement study of efficiency for llm applications on mobile devices. In *Proceedings of the 2nd International Workshop on Foundation Models for Cyber-Physical Systems & Internet of Things*, pp. 19–24, 2025.

An Yang, Baosong Yang, Beichen Zhang, Binyuan Hui, Bo Zheng, Bowen Yu, Chengyuan Li, Dayiheng Liu, Fei Huang, Haoran Wei, Huan Lin, Jian Yang, Jianhong Tu, Jianwei Zhang, Jianxin Yang, Jiaxi Yang, Jingren Zhou, Junyang Lin, Kai Dang, Keming Lu, Keqin Bao, Kexin Yang, Le Yu, Mei Li, Mingfeng Xue, Pei Zhang, Qin Zhu, Rui Men, Runji Lin, Tianhao Li, Tianyi Tang, Tingyu Xia, Xingzhang Ren, Xuancheng Ren, Yang Fan, Yang Su, Yichang Zhang, Yu Wan, Yuqiong Liu, Zeyu Cui, Zhenru Zhang, and Zihan Qiu. Qwen2.5 technical report, 2025. URL https://arxiv.org/abs/2412.15115.

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*, pp. 4791–4800, 2019.

Yue Zheng, Yuhao Chen, Bin Qian, Xiufang Shi, Yuanchao Shu, and Jiming Chen. A review on edge large language models: Design, execution, and applications. *ACM Computing Surveys*, 57(8):1–35, 2025.

Zhaojing Zhou, Xunchao Li, Minghao Li, Handi Zhang, Haoshuang Wang, Wenbin Chang, Yiqun Liu, Qingqing Dang, Dianhai Yu, Yanjun Ma, et al. Ccq: Convolutional code for extreme low-bit quantization in llms. *arXiv preprint arXiv:2507.07145*, 2025.

A NOTATION SUMMARY

A summary of the notation used in our practical LLM weight-quantization setup is provided in Table 2 for reference and convenience of the reader. Matrices are denoted by bold uppercase letters (e.g., $\boldsymbol{W}$), vectors by bold lowercase letters (e.g., $\boldsymbol{x}$), and scalars by lowercase letters (e.g., x). Random variables are represented using sans-serif font (e.g., W), while deterministic quantities use the standard math font.

Table 2: Notation summary

Symbol	Description
$\mathsf{W} \in \mathbb{R}^{d_{\text{out}} \times d_{\text{in}}}$	Weight matrix of a linear layer (Random variable)
$\widehat{\mathsf{W}}$	Quantized reconstruction of W (Random variable)
$\boldsymbol{X} \in \mathbb{R}^{d_{\text{in}} \times n}$	Representative input data matrix
$d_{\text{WMSE}}(\mathsf{W}, \widehat{\mathsf{W}})$	Weighted mean squared reconstruction error (WMSE)
$L_{\text{WMSE}}(\mathsf{W}, \widehat{\mathsf{W}})$	Normalized, single-realization WMSE training loss
$\mathsf{Y} = \mathsf{W}\boldsymbol{X} \in \mathbb{R}^{d_{\text{out}} \times n}$	Layer output activations (Random variable)
$\mathsf{y} = \text{vec}(\mathsf{Y}) \in \mathbb{R}^{d_{\text{out}} \cdot n}$	Vectorized layer output (Random variable)
$\mathsf{w} = \text{vec}(\mathsf{W}) \in \mathbb{R}^{d_{\text{out}} \cdot d_{\text{in}}}$	Vectorized weight matrix (Random variable)
$\widehat{\mathsf{W}}^{(k)}$	Reconstruction of W at refinement stage k
$\widehat{\mathsf{Y}}^{(k)}$	Output produced by the k-stage reconstruction
D_k	Distortion level at refinement stage k
Σ_Y	Covariance of vectorized output Y
λ_i	Eigenvalues of Σ_Y in Gaussian eigen-decomposition
v_i	i-th scalar Gaussian component in the eigen-basis ($\mathsf{v}_i \sim \mathcal{N}(0, \lambda_i)$)
$d_{\text{in}}, d_{\text{out}}$	Input and output dimensions of the layer
n	Number of calibration samples
g	Group size for partitioning weights
M	Number of additive codebooks
$C_m \in \mathbb{R}^{g \times 2^B}$	m-th codebook containing 2^B codewords
B	Number of bits used to index codewords
$b_{ijm} \in \{0,1\}^{2^B}$	One-hot vector selecting a codeword from codebook m
k	Number of active codebooks at inference ($k \leq M$)
$L^{(k)}$	Drop-by-Drop loss at refinement stage k
α_k	Matryoshka loss coefficient for refinement stage k

B AQLM'S OPTIMIZATION PROCESS

The quantization process in AQLM learns both the codebooks C and the assignments b, where b encodes which codewords from each codebook are selected to approximate groups of weights. Initialization begins with a residual K-means procedure. Firstly, a standard K-means clustering is applied to the grouped weights to generate initial cluster centers and assignments (Chen et al., 2010). Then, the quantization errors are computed as the differences between weights and their assigned centroids. Lastly, subsequent codebooks are iteratively initialized by clustering these *residual* errors, enabling each additional codebook to compensate for errors left by the previous ones. This hierarchical initialization helps achieve an effective residual-based representation.

Learning proceeds through iterative alternating optimization of the discrete codes and continuous codebooks, consisting of three main phases:

1. **Code Optimization:** The discrete assignment vectors b are refined using a beam search algorithm, which searches for combinations of codewords that minimize the quantization error between the original and quantized model outputs. This step focuses on minimizing the error in model outputs induced by quantization.
2. **Codebook Update:** Given fixed codes, the codebooks C are updated by solving a least squares problem that minimizes the difference between the original and quantized model outputs. This gradient-based step optimizes the continuous parameters of the codebooks to better fit the data.
3. **Intra-Layer Fine-Tuning:** Finally, to further reduce residual quantization errors, AQLM performs fine-tuning at the transformer block level. This phase *jointly* adjusts the non-quantized model parameters and the codebooks, ensuring that the outputs of the quantized model closely match those of the original (Chen & Khisti, 2024).

Estimating Model Size. To evaluate the practical memory footprint of an AQLM-quantized model, we estimate the total storage cost for a given codebook configuration. The memory required for a quantized weight matrix comprises three primary components: the codebooks, the discrete codes, and the per-unit scales.

Specifically, for a weight matrix with input dimension d_{in}, output dimension d_{out}, and group size g, using M codebooks corresponding to B-bit codes, the total amount of memory (in bits) is calculated as follows. The codebooks and scales are stored using standard 16-bit floating-point precision (FP16):

- **Codebooks:** $g \cdot M \cdot 2^B \cdot 16$
- **Codes:** $d_{out} \cdot (d_{in}/g) \cdot B \cdot M$
- **Scales:** $d_{out} \cdot 16$

The *average bits per parameter* ($\bar{b}$) is then computed by dividing the total size in bits by the total number of parameters:

$$\bar{b} = \frac{\text{size in bits}}{\text{number of parameters}} = \frac{16 * g * M * 2^B + d_{out} * (d_{in}/g) * B * M + 16 * d_{out}}{d_{out} * d_{in}} \tag{2}$$

As a concrete example, consider the `mlp.gate_proj` layer of a Llama 2 70B model ($d_{in} = 8192$, $d_{out} = 28672$). With a group size $g = 8$ and two 8-bit codebooks ($M = 2, B = 8$), Equation 2 yields approximately 2.002 bits per parameter. Typically, the storage cost is dominated by the codes, while the codebooks and scales induce a relatively small memory overhead (Egiazarian et al., 2024).

C Empirical Validation of the Gaussian Source Assumption

C.1 Weight Distribution Visualization

Figure 9 shows distributions of LLM weight parameters from the first three transformer layers of Qwen2.5-7B-Instruct. Each layer contains seven weight matrices - four attention and three multi-layer perceptron (MLP) projections.

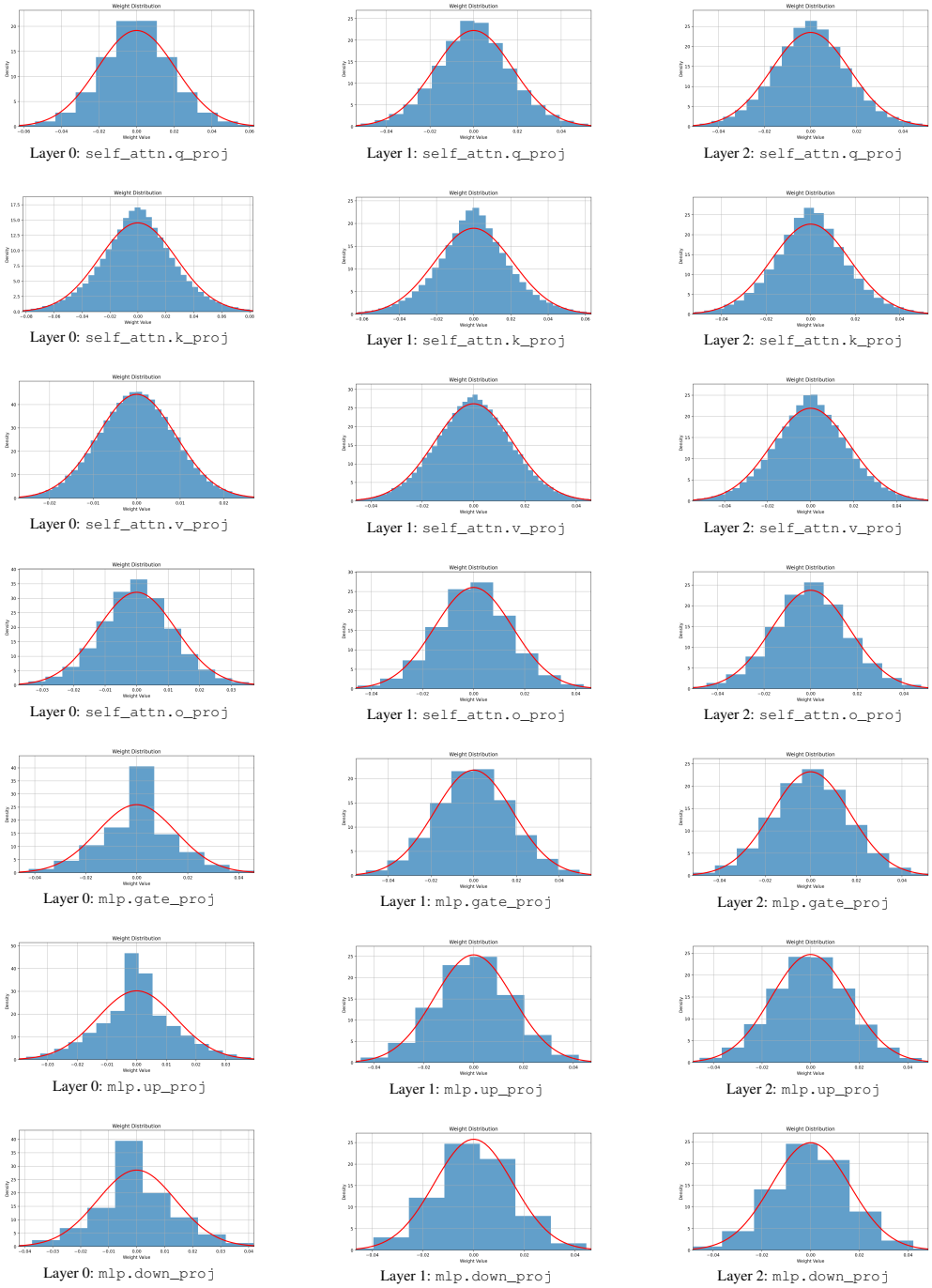


Figure 9: Qwen2.5-7B-Instruct weights fitted to Gaussian distributions

C.2 KURTOSIS ANALYSIS AND WALSH-HADAMARD ROTATIONS

In Section 4, we established that a Gaussian source is successively refinable under WMSE distortion. In practice, however, LLM weight distributions are leptokurtic: they have heavier tails than a Gaussian, with excess kurtosis values ranging from $+0.7$ to $+6.1$ across projection types, as shown in Table 3. The closer the weight distribution is to Gaussian, the more tightly the successive refinement guarantee applies, which motivates pre-processing the weights before quantization.

Prior work has shown that applying a random orthogonal transformation, particularly a Hadamard transform, to LLM weight matrices redistributes weight magnitude uniformly across coordinates, enforcing a better match between the processed weight distribution and the Gaussian distribution Ashkboos et al. (2024); Tseng et al. (2024); Chee et al. (2023); Malinovskii et al. (2025); Lee & Song (2026). Motivated by our theoretical result that Gaussian sources are successively refinable under WMSE, we incorporate a block-wise Walsh-Hadamard rotation into the Drop-by-Drop pipeline as an optional pre-processing step, extending this technique to the multi-bitwidth additive codebook setting. Table 3 confirms the effect: model-average excess kurtosis drops across all five QWEN model sizes.

Table 3: Excess kurtosis of raw and block-Hadamard-rotated weights across the Qwen model family, measured per projection type. By definition, a pure Gaussian has excess kurtosis of 0. Rotation consistently reduces leptokurtosis across all models and projections, bringing the weight distributions closer to the Gaussian source assumed by our theory.

Projection	**Qwen-0.5B**		**Qwen-3B**		**Qwen-7B**	
	Raw	Rotated	Raw	Rotated	Raw	Rotated
q_proj	+5.5267	+2.6913	+1.9793	+1.2025	+2.0034	+1.0335
k_proj	+3.1480	+1.5132	+2.0037	+1.0912	+2.4605	+1.0302
v_proj	+1.9380	+0.1212	+0.9248	+0.0675	+1.1541	+0.1662
o_proj	+6.0979	+1.6146	+1.8030	+0.4334	+3.2082	+0.5911
gate_proj	+1.9406	+0.7544	+2.1402	+1.5026	+5.5342	+0.2676
up_proj	+1.8242	+0.7688	+2.6955	+1.9819	+0.7627	+0.2531
down_proj	+1.7905	+0.3604	+2.9241	+0.2324	+2.2574	+0.1585
Model Avg	**+3.1809**	**+1.1177**	**+2.0672**	**+0.9302**	**+2.4829**	**+0.5000**

Projection	**Qwen-14B**		**Qwen-32B**	
	Raw	Rotated	Raw	Rotated
q_proj	+3.0286	+0.5989	+2.5299	+0.5457
k_proj	+2.3982	+0.7719	+2.2827	+0.7528
v_proj	+0.7241	+0.1547	+0.7523	+0.1479
o_proj	+1.6667	+0.4958	+1.5147	+0.4745
gate_proj	+1.9359	+0.9059	+1.9455	+0.6978
up_proj	+2.1167	+1.0816	+2.7827	+1.0451
down_proj	+2.8015	+0.6878	+2.6904	+0.5914
Model Avg	**+2.0960**	**+0.6709**	**+2.0712**	**+0.6079**

D THEORETICAL FOUNDATIONS AND SUCCESSIVE REFINEMENT

D.1 DEFINITION OF SUCCESSIVE REFINEMENT

The successive refinement problem, introduced by Equitz & Cover (1991), asks whether a source (W, d) can be encoded in stages with no rate loss. Specifically, for any finite distortion chain $D_1 > D_2 > \cdots > D_K > 0$, we ask whether there exists a single embedded code whose k-prefix achieves the optimal rate-distortion pair $(R(D_k), D_k)$ for all $k = 1, \ldots, K$, with no rate penalty compared to encoding directly at D_k. For target distortions $D_1 > D_2$, this means encoding W^N first at rate R_1 with distortion D_1, then refining by an additional rate $R_2 - R_1$ to reach distortion D_2, without exceeding the total rate $R(D_2)$. The following definition formalizes this notion.

Definition 1 (Successive Refinability of General Sources)**.** *A source is said to be successively refinable from D_1 to D_2 if there exists a sequence of encoders*

$$i_N : \mathcal{W}^N \to \{1, \ldots, 2^{NR_1}\}, \qquad j_N : \mathcal{W}^N \to \{1, \ldots, 2^{N(R_2-R_1)}\},$$

and respective reconstruction functions (decoders)

$$g_{1N} : \{1, \ldots, 2^{NR_1}\} \to \widehat{\mathcal{W}}_1^N, \qquad g_{2N} : \{1, \ldots, 2^{NR_1}\} \times \{1, \ldots, 2^{N(R_2-R_1)}\} \to \widehat{\mathcal{W}}_2^N.$$

such that the reconstructions

$$\widehat{\mathsf{W}}_1^N = g_{1N}\big(i_N(\mathsf{W}^N)\big), \qquad \widehat{\mathsf{W}}_2^N = g_{2N}\big(i_N(\mathsf{W}^N),\, j_N(\mathsf{W}^N)\big),$$

satisfy

$$\limsup_{N\to\infty} \mathbb{E}\Big[d(\mathsf{W}^N, \widehat{\mathsf{W}}_1^N)\Big] \le D(R_1) + \varepsilon_N, \qquad \limsup_{N\to\infty} \mathbb{E}\Big[d(\mathsf{W}^N, \widehat{\mathsf{W}}_2^N)\Big] \le D(R_2) + \varepsilon_N,$$

where $\varepsilon_N \to 0$ as $N \to \infty$ and $D(R)$ is the distortion–rate function. Then,

$$R_1 = R(D_1) + \varepsilon_1, \qquad R_{2|1} = R(D_2) - R(D_1) + \varepsilon_2,$$

and hence the total rate satisfies

$$R_{\textit{total}} = R_1 + R_{2|1} = R(D_2) + \varepsilon_1 + \varepsilon_2,$$

where $\varepsilon_1, \varepsilon_2 > 0$ are arbitrarily small constants. Note that $R(D_1)$ and $R(D_2)$ are theoretical rate–distortion limits, whereas R_1, $R_{2|1}$, and $R_{\textit{total}}$ are empirical operational rates of the actual coding scheme. Here, the rate in stages matches the single-shot rate required to achieve distortion D_2. Thus, encoding in two stages (first at rate R_1 and then refining by an additional $R_{2|1}$) achieves the same operational point as single-shot encoding at rate $R(D_2)$, with no rate penalty. More formally, the point $(R_1, R_2) = (R(D_1), R(D_2))$ is asymptotically achievable by a two-stage description in which the second stage refines the first. Generally, a problem defined by a source distribution $p(\mathsf{w})$ and distortion measure $d(w, \widehat{w})$ is said to be successively refinable, if successive refinement from distortion D_1 to D_2 is achievable for every $D_1 \ge D_2$.

D.2 CLASSICAL RESULT: GAUSSIAN SOURCES ARE SUCCESSIVELY REFINABLE

Next, we show that Gaussian sources are successively refinable. We first establish the single-source case in Theorem 2.

Theorem 2 (Successive Refinability of Gaussian Sources)**.** *A Gaussian source $\mathsf{w} \sim \mathcal{N}(0, \sigma^2)$ under the mean squared error (MSE) distortion measure $d(w, \widehat{w}) = (w - \widehat{w})^2$ is successively refinable Equitz (1989).*

Since the Gaussian source achieves its rate-distortion function by a single-letter test channel, the N-block problem separates into N independent scalar problems. For notational clarity, we therefore work with the single-letter random variables w, $\widehat{\mathsf{w}}^{(1)}$, and $\widehat{\mathsf{w}}^{(2)}$. The superscripts denote refinement stages.

The Gaussian rate-distortion function is

$$R(D) = \frac{1}{2} \log \frac{\sigma^2}{D}, \qquad 0 < D \le \sigma^2.$$

The scalar Gaussian test channel induces the Markov chain $\widehat{\mathsf{w}}^{(1)} \to \widehat{\mathsf{w}}^{(2)} \to \mathsf{w}$, which ensures that each refinement layer depends only on its predecessor. Concretely, define mutually independent Gaussian increments

$$\mathsf{z}^{(1)} \sim \mathcal{N}(0, \sigma^2 - D_1), \qquad \mathsf{z}^{(2)} \sim \mathcal{N}(0, D_1 - D_2),$$

and construct the successive reconstructions by adding these increments,

$$\widehat{\mathsf{w}}^{(0)} = 0, \qquad \widehat{\mathsf{w}}^{(k)} = \widehat{\mathsf{w}}^{(k-1)} + \mathsf{z}^{(k)}, \qquad k = 1, 2.$$

Furthermore, the additive Gaussian construction satisfies

$$I(\mathsf{w}; \widehat{\mathsf{w}}^{(1)}) = R(D_1), \qquad I(\mathsf{w}; \widehat{\mathsf{w}}^{(1)}, \widehat{\mathsf{w}}^{(2)}) = R(D_2).$$

Realizing this single-letter distribution operationally requires coding over blocks of length N. Construction of the code is below, as we work with the block sequences $\mathsf{w}^N, \widehat{\mathsf{w}}^{(1)N}, \widehat{\mathsf{w}}^{(2)N}$.

Let $\mathcal{C}_1$ represent the first codebook, consisting of the codewords

$$\mathcal{C}_1 := \big\{\widehat{\mathsf{w}}^{(1)N}(m_1)\big\}_{m_1=1}^{2^{NR_1}}, \qquad m_1 \in \{1, \dots, 2^{NR_1}\},$$

with each codeword drawn i.i.d., $\mathcal{N}(0, \sigma^2 - D_1)$.

Similarly, for each m_1, let $\mathcal{C}_2(m_1)$ represent the second codebook, consisting of the codewords

$$\mathcal{C}_2(m_1) := \big\{\widehat{\mathsf{w}}^{(2)N}(m_1, m_2)\big\}_{m_2=1}^{2^{NR_{2|1}}}, \qquad m_2 \in \{1, \dots, 2^{NR_{2|1}}\},$$

with each codeword drawn i.i.d. $\widehat{\mathsf{w}}^{(2)N}(m_1, m_2) = \widehat{\mathsf{w}}^{(1)N}(m_1) + \mathsf{z}^{(2)N}$, $\mathsf{z}^{(2)} \sim \mathcal{N}(0, D_1 - D_2)$.

The encoder i_N selects an index of a codeword in $\mathcal{C}_1$ that is the closest reconstruction to w^N,

$$i_N(\mathsf{w}^N) := \operatorname*{arg\,min}_{m_1 \in \{1, \dots, 2^{NR_1}\}} d\big(\mathsf{w}^N, \widehat{\mathsf{w}}^{(1)N}(m_1)\big),$$

and the decoder g_{1N} outputs the codeword at that selected index,

$$g_{1N}\big(i_N(\mathsf{w}^N)\big) = \widehat{\mathsf{w}}^{(1)N}\big(i_N(\mathsf{w}^N)\big).$$

Given the selected index $i_N(\mathsf{w}^N)$, the encoder j_N selects an index of a codeword in $\mathcal{C}_2(i_N(\mathsf{w}^N))$ that is the closest reconstruction to w^N,

$$j_N(\mathsf{w}^N) := \operatorname*{arg\,min}_{m_2 \in \{1, \dots, 2^{NR_{2|1}}\}} d\Big(\mathsf{w}^N, \widehat{\mathsf{w}}^{(2)N}\big(i_N(\mathsf{w}^N), m_2\big)\Big),$$

and, likewise, given both selected indices, the decoder g_{2N} outputs

$$g_{2N}\big(i_N(\mathsf{w}^N), j_N(\mathsf{w}^N)\big) = \widehat{\mathsf{w}}^{(2)N}\big(i_N(\mathsf{w}^N), j_N(\mathsf{w}^N)\big).$$

Following Definition 1, for any distortion pair $D_1 > D_2 > 0$, the constructed encoders and decoders achieve

$$R_1 = R(D_1) + \varepsilon_1, \qquad R_{2|1} = R(D_2) - R(D_1) + \varepsilon_2,$$

and hence a total rate

$$R_{\text{total}} = R_1 + R_{2|1} = R(D_2) + \varepsilon_1 + \varepsilon_2,$$

where $\varepsilon_1, \varepsilon_2 > 0$ can be made arbitrarily small.

Hence, the achievable region coincides exactly with the single-shot rate-distortion limits. In other words, no additional rate is required when encoding the Gaussian source in successive refinement stages compared to encoding it once at distortion D_2. This establishes that Gaussian sources under MSE distortion are successively refinable.

Achievability. Following the approach by Cover & Thomas (Cover & Thomas, 2006, Section 10.5), the achievability is established using standard typicality arguments via the Asymptotic Equipartition Property (AEP). Let $\varepsilon_1, \varepsilon_2 > 0$, and let typicality penalties $0 < \delta_1 < \varepsilon_1$ and $0 < \delta_2 < \varepsilon_2$.

Generation of codebooks. As constructed above, the first codebook $\mathcal{C}_1$ consists of 2^{NR_1} codewords of length N, where each symbol of every codeword is drawn i.i.d. following $\mathcal{N}(0, \sigma^2 - D_1)$. For each m_1, the second codebook $\mathcal{C}_2(m_1)$ consists of $2^{NR_{2|1}}$ codewords, built by taking the base codeword $\widehat{\mathsf{w}}^{(1)N}(m_1)$ and adding an independent sequence of Gaussian increments $\mathsf{z}^{(2)N}$. Reveal these codebooks to the encoder and the decoder.

Encoding. The encoder first looks for an index m_1 such that $\big(\mathsf{w}^N, \widehat{\mathsf{w}}^{(1)N}(m_1)\big) \in A_{d,\delta_1}^{(N)}$, the first-stage distortion typical set. If there is more than one such m_1, send the least. If there is no such m_1, let $m_1 = 1$. The chosen index is denoted as $i_N(\mathsf{w}^N)$. Similar steps are followed to select the second-stage index $j_N(\mathsf{w}^N) = m_2$ from the refinement codebook $\mathcal{C}_2(i_N(\mathsf{w}^N))$ such that the sequences fall within the second-stage distortion typical set $A_{d,\delta_2}^{(N)}$.

Decoding. g_{1N}, g_{2N} output the selected codewords.

Calculation. Let $A_\gamma^{(N)}$ denote the typical set of the source for some tolerance $\gamma > 0$. By the joint AEP, for any typical source sequence $\mathsf{w}^N \in A_\gamma^{(N)}$, the probability that a randomly drawn codeword falls within the joint distortion typical set $A_{d,\delta_1}^{(N)}$ satisfies

$$\Pr\left[\big(\mathsf{w}^N, \widehat{\mathsf{w}}^{(1)N}(m_1)\big) \in A_{d,\delta_1}^{(N)}\right] \geq 2^{-N(R(D_1)+\delta_1)}$$

for N sufficiently large. Let $P_e^{(1)}$ denote the probability that *no* codeword in $\mathcal{C}_1$ falls within this first-stage distortion typical set. We bound this by the probability that the source is atypical, plus the probability that the codebook fails for a typical source:

$$P_e^{(1)} \leq \Pr\big(\mathsf{w}^N \notin A_\gamma^{(N)}\big) + \Big(1 - 2^{-N(R(D_1)+\delta_1)}\Big)^{2^{N(R(D_1)+\varepsilon_1)}}.$$

By the standard AEP, the first term $\Pr\big(\mathsf{w}^N \notin A_\gamma^{(N)}\big) \to 0$. For the second term, since $\delta_1 < \varepsilon_1$,

$$\Big(1 - 2^{-N(R(D_1)+\delta_1)}\Big)^{2^{N(R(D_1)+\varepsilon_1)}} \leq \exp\Big(-2^{N(\varepsilon_1-\delta_1)}\Big) \xrightarrow[N\to\infty]{} 0.$$

Therefore, the total probability of encoding error $P_e^{(1)} \to 0$. Conditioned on the event that $\mathcal{C}_1$ succeeds, the identical argument applied to $\mathcal{C}_2(i_N(\mathsf{w}^N))$ using the respective typicality bound $\delta_2 < \varepsilon_2$ gives $P_e^{(2)} \to 0$.

On the event that both codebooks succeed (which occurs with probability $\geq 1 - P_e^{(1)} - P_e^{(2)} \to 1$), the reconstructions satisfy $d(\mathsf{w}^N, \widehat{\mathsf{w}}^{(1)N}) \leq D_1$ and $d(\mathsf{w}^N, \widehat{\mathsf{w}}^{(2)N}) \leq D_2$ by construction. Hence

$$\mathbb{E}\big[d(\mathsf{w}^N, \widehat{\mathsf{w}}^{(1)N})\big] \to D_1, \qquad \mathbb{E}\big[d(\mathsf{w}^N, \widehat{\mathsf{w}}^{(2)N})\big] \to D_2,$$

so the reconstructions $\widehat{\mathsf{W}}_1^N = g_{1N}(i_N(\mathsf{w}^N))$ and $\widehat{\mathsf{W}}_2^N = g_{2N}(i_N(\mathsf{w}^N), j_N(\mathsf{w}^N))$ satisfy the distortion bounds of Definition 1 at rates $R_1 = R(D_1) + \varepsilon_1$ and $R_{2|1} = R(D_2) - R(D_1) + \varepsilon_2$.

D.3 Definition of Successive Refinement for Vector Sources

Next, we extend this successive refinability result to vector Gaussian sources.

Definition 2 (Successive Refinability of Vector Sources)**.** *A vector source $\mathbf{w} = (\mathsf{w}_1, \ldots, \mathsf{w}_m) \in \mathcal{W}^m$ with independent components is said to be successively refinable across a distortion chain $D^{(1)} > D^{(2)} > \cdots > D^{(K)}$ if there exists a sequence of encoders*

$$i_N^{(1)} : (\mathcal{W}^m)^N \to \{1, \ldots, 2^{NR^{(1)}}\}, \qquad i_N^{(k)} : (\mathcal{W}^m)^N \to \{1, \ldots, 2^{N(R^{(k)} - R^{(k-1)})}\}, \quad k = 2, \ldots, K,$$

and respective reconstruction functions (decoders)

$$g_N^{(k)} : \{1, \ldots, 2^{NR^{(1)}}\} \times \cdots \times \{1, \ldots, 2^{N(R^{(k)} - R^{(k-1)})}\} \to \big(\widehat{\mathcal{W}}_k^m\big)^N, \qquad k = 1, \ldots, K,$$

such that the stage-k reconstructions

$$\widehat{\mathbf{w}}^{(k)N} = g_N^{(k)}\big(i_N^{(1)}(\mathbf{w}^N), \ldots, i_N^{(k)}(\mathbf{w}^N)\big)$$

satisfy

$$\limsup_{N\to\infty} \mathbb{E}\big[d(\mathbf{w}^N, \widehat{\mathbf{w}}^{(k)N})\big] \le D^{(k)} + \varepsilon_N^{(k)}, \qquad k = 1, \ldots, K,$$

where $\varepsilon_N^{(k)} \to 0$ as $N \to \infty$ and our distortion measure is the mean squared error, $d(\boldsymbol{w}, \widehat{\boldsymbol{w}}) = \sum_{i=1}^m (w_i - \widehat{w}_i)^2$. The cumulative rate through stage k is $R^{(k)} = R(D^{(k)}) + \sum_{j=1}^k \varepsilon_j$, where the rate excesses $\varepsilon_j > 0$ are arbitrarily small constants. A vector source is said to be successively refinable, if successive refinement is achievable across the distortion chain.

D.4 Gaussian Vector Sources are Successively Refinable

The original construction of Equitz (1989) considers a single Gaussian source $\mathsf{w} \sim \mathcal{N}(0, \sigma^2)$. We build the analogous construction component-wise for an independent Gaussian vector source

$$\mathsf{w}_i \sim \mathcal{N}(0, \sigma_i^2), \qquad i = 1, \ldots, m.$$

Theorem 3 (Successive Refinability of Gaussian Vector Sources)**.** *Let $\mathbf{w} = (\mathsf{w}_1, \ldots, \mathsf{w}_m)$ be a vector of independent Gaussian sources $\mathsf{w}_i \sim \mathcal{N}(0, \sigma_i^2)$, $i = 1, \ldots, m$, under the mean squared error distortion measure $d(\boldsymbol{w}, \widehat{\boldsymbol{w}}) = \sum_{i=1}^m (w_i - \widehat{w}_i)^2$. Then $\mathbf{w}$ is successively refinable across any number of stages K.*

For each stage $k = 1, \ldots, K$, the per-component distortion targets are chosen via reverse water-filling,

$$D_i^{(k)} \;=\; \min\big(\sigma_i^2,\, \mu_k\big)\,, \qquad D_i^{(0)} \;\coloneqq\; \sigma_i^2,$$

where $\mu_1 > \mu_2 > \cdots > \mu_K > 0$ is a strictly decreasing sequence of water levels chosen so that $\sum_i D_i^{(k)} = D^{(k)}$ at each stage. This is the classical solution to the parallel Gaussian rate-distortion problem (Cover & Thomas, 2006, Ch. 10), guaranteeing that $R^{(k)} = \sum_i R_i^{(k)}$ equals the true rate-distortion function $R(D^{(k)})$ at each stage. This gives, in particular, a nested chain of distortion targets for each component,

$$D_i^{(0)} = \sigma_i^2 \;\ge\; D_i^{(1)} \;\ge\; \cdots \;\ge\; D_i^{(K)} \;>\; 0,$$

Define mutually independent Gaussian increments

$$\mathsf{z}_i^{(k)} \sim \mathcal{N}\Big(0,\, D_i^{(k-1)} - D_i^{(k)}\Big)\,, \qquad k = 1, \ldots, K,$$

and construct the successive reconstructions by accumulating these increments,

$$\widehat{\mathsf{w}}_i^{(0)} = 0, \qquad \widehat{\mathsf{w}}_i^{(k)} = \widehat{\mathsf{w}}_i^{(k-1)} + \mathsf{z}_i^{(k)}, \qquad k = 1, \ldots, K.$$

By construction, each stage lands exactly on its target distortion,

$$\mathbb{E}\Big[\big(\mathsf{w}_i - \widehat{\mathsf{w}}_i^{(k)}\big)^2\Big] = D_i^{(k)}, \qquad k = 1, \ldots, K.$$

Collecting components into vectors makes the joint structure explicit: let $\mathbf{w} = (\mathsf{w}_1, \ldots, \mathsf{w}_m)^\top$, $\widehat{\mathbf{w}}^{(k)} = (\widehat{\mathsf{w}}_1^{(k)}, \ldots, \widehat{\mathsf{w}}_m^{(k)})^\top$, and $\boldsymbol{D}^{(k)} = \mathrm{diag}(D_1^{(k)}, \ldots, D_m^{(k)})$, with $\boldsymbol{D}^{(0)} = \mathrm{diag}(\sigma_1^2, \ldots, \sigma_m^2)$. Then

$$\widehat{\mathbf{w}}^{(k)} = \widehat{\mathbf{w}}^{(k-1)} + \mathbf{z}^{(k)}, \qquad \mathbf{z}^{(k)} \sim \mathcal{N}\big(\mathbf{0},\, \boldsymbol{D}^{(k-1)} - \boldsymbol{D}^{(k)}\big), \quad k = 1, \ldots, K,$$

where $\mathbf{z}^{(k)} \perp \widehat{\mathbf{w}}^{(k-1)}$. Hence, the Markov chain essential for successive refinability holds by construction:

$$\widehat{\mathbf{w}}^{(1)} \;\to\; \widehat{\mathbf{w}}^{(2)} \;\to\; \cdots \;\to\; \widehat{\mathbf{w}}^{(K)} \;\to\; \mathbf{w}.$$

Achievable Rates. The achievable rate required to reach distortion $D_i^{(k)}$ for component i, encoded from scratch, is

$$R_i^{(k)} = \frac{1}{2} \log \frac{\sigma_i^2}{D_i^{(k)}}, \qquad k = 1, \ldots, K,$$

and for the m-component vector, $R^{(k)} = \sum_{i=1}^{m} R_i^{(k)}$. Under the staged construction, the rate spent at stage k is the mutual information between w_i and its new reconstruction, given the reconstruction from the previous stage:

$$I\big(\mathsf{w}_i;\, \widehat{\mathsf{w}}_i^{(k)} \mid \widehat{\mathsf{w}}_i^{(k-1)}\big) = h\big(\mathsf{w}_i \mid \widehat{\mathsf{w}}_i^{(k-1)}\big) - h\big(\mathsf{w}_i \mid \widehat{\mathsf{w}}_i^{(k-1)}, \widehat{\mathsf{w}}_i^{(k)}\big) = \tfrac{1}{2}\log\frac{D_i^{(k-1)}}{D_i^{(k)}},$$

and summing over components gives the total incremental rate at stage k,

$$\Delta R^{(k)} = \frac{1}{2}\sum_{i=1}^{m}\log\frac{D_i^{(k-1)}}{D_i^{(k)}} = R^{(k)} - R^{(k-1)}, \qquad k = 1, \dots, K.$$

Single-Shot Encoding Equivalence. Encoding directly to any distortion level $D^{(k)}$, without passing through the intermediate stages, requires rate $R^{(k)} = \frac{1}{2}\sum_{i=1}^{m}\log\frac{\sigma_i^2}{D_i^{(k)}}$. Summing the incremental rates $\Delta R^{(j)}$ from stage 1 through stage k shows this same total is reached by the staged construction, since each intermediate distortion level cancels against itself:

$$\begin{aligned}
\sum_{j=1}^{k}\Delta R^{(j)} &= \frac{1}{2}\sum_{i=1}^{m}\left[\log\frac{D_i^{(0)}}{D_i^{(1)}} + \log\frac{D_i^{(1)}}{D_i^{(2)}} + \cdots + \log\frac{D_i^{(k-1)}}{D_i^{(k)}}\right] \\
&= \frac{1}{2}\sum_{i=1}^{m}\log\frac{D_i^{(0)}}{D_i^{(k)}} \\
&= \frac{1}{2}\sum_{i=1}^{m}\log\frac{\sigma_i^2}{D_i^{(k)}} \\
&= R^{(k)}, \qquad k = 1, \dots, K.
\end{aligned}$$

Hence there is no rate penalty for encoding in K successive stages rather than in a single shot at any target distortion level.

Achievability. The achievability is established by using AEP to the parallel scalar channels. Let $\varepsilon_{k,i} > 0$ and let typicality penalties $0 < \delta_{k,i} < \varepsilon_{k,i}$ for each component i at refinement stage k.

Generation of codebooks. For each active component i at stage $k = 1$, generate an independent codebook $\mathcal{C}_{1,i}$ consisting of $2^{NR_{1,i}}$ codewords drawn i.i.d. following $\mathcal{N}(0, \sigma_i^2 - D_i^{(1)})$, where $R_{1,i} = \frac{1}{2}\log(\sigma_i^2/D_i^{(1)}) + \varepsilon_{1,i}$. For successive stages $k > 1$, for each active component i, generate additional codebooks consisting of $2^{N(R_{k,i} - R_{k-1,i})}$ additive Gaussian increments drawn i.i.d. as $\mathcal{N}(0, D_i^{(k-1)} - D_i^{(k)})$. Reveal these codebooks to the encoder and the decoder.

Encoding. At stage k, for each active component i, the encoder independently looks for a codeword index $m_{k,i}$ such that the sequence and its reconstruction fall within the scalar distortion typical set $A_{d,\delta_{k,i}}^{(N)}$. If there is more than one such index, send the least. If there is no such index, default to the first index. The total stage-k index transmitted is the concatenation of these parallel indices, with a total incremental rate $\Delta R^{(k)} = \sum_{i=1}^{m}(R_{k,i} - R_{k-1,i})$.

Decoding. The decoders receive the concatenated indices, separate them by component, and output the vector sequence composed of the individual reconstructions.

Calculation. Similar to AEP used in Theorem 2, the probability $P_{e,i}^{(k)}$ that a specific component i fails to find a typical codeword at stage k is bounded by:

$$P_{e,i}^{(k)} \le \exp\Big(-2^{N(\varepsilon_{k,i} - \delta_{k,i})}\Big) \xrightarrow[N\to\infty]{} 0.$$

Because the vector encoder fails if at least one of the active components fails, we apply a union bound over the m independent components. Let $P_{e,\text{total}}^{(k)}$ be the total probability of an encoding error at stage k:

$$P_{e,\text{total}}^{(k)} \le \sum_{i=1}^{m} P_{e,i}^{(k)} \xrightarrow[N\to\infty]{} 0.$$

Because m is finite and each individual error probability vanishes double-exponentially, the union bound strictly goes to 0. When all parallel codebooks succeed, the vector expectation is simply the sum of the scalar expectations. Hence,

$$\mathbb{E}\big[d(\mathbf{w}^N, \widehat{\mathbf{w}}^{(k)N})\big] = \frac{1}{N}\sum_{i=1}^{m} \mathbb{E}\big[(\mathsf{w}_i^N - \widehat{\mathsf{w}}_i^{(k)N})^2\big] \to \sum_{i=1}^{m} D_i^{(k)} = D^{(k)}.$$

The achievable region matches the single-shot rate-distortion bounds with no rate penalty, hence, completing the proof.

This construction reappears later: after the diagonalization of Σ_Y in Theorem 4, we obtain independent scalar Gaussian components with eigenvalues λ_i in place of σ_i^2, and apply the same per-component test channels to prove Theorem 5.

E Proofs of Auxiliary Results

E.1 Rate-Distortion Under Weighted MSE

The classical rate-distortion function for Gaussian sources under MSE is solved by reverse water-filling (Cover & Thomas, 2006, Ch. 10). Sakrison Sakrison (1968) extended this to weighted square error for continuous-time processes using the Karhunen-Loève expansion. The following theorem adapts this result to the finite-dimensional matrix setting relevant to LLM quantization, replacing the Karhunen-Loève expansion with a standard eigen-decomposition. Then, it extends it to all structural cases of $\boldsymbol{X}$, which is necessary since the calibration matrix in LLM quantization is typically wide ($d_{\text{in}} < n$).

Theorem 4 (WMSE Rate-Distortion Equivalence)**.** *Let $\mathbf{W} \in \mathbb{R}^{d_{\text{out}} \times d_{\text{in}}}$ be a zero-mean i.i.d. Gaussian random matrix, and let $\boldsymbol{X} \in \mathbb{R}^{d_{\text{in}} \times n}$ be a deterministic matrix. Consider the weighted MSE distortion*

$$d_{\text{WMSE}}(\mathbf{W}, \widehat{\mathbf{W}}) = \mathbb{E}\Big[\big\|(\mathbf{W} - \widehat{\mathbf{W}})\,\boldsymbol{X}\big\|_F^2\Big].$$

Then, for the transformed source $\mathbf{Y} = \mathbf{W}\boldsymbol{X} \in \mathbb{R}^{d_{\text{out}} \times n}$,

$$d_{\text{WMSE}}(\mathbf{W}, \widehat{\mathbf{W}}) \;=\; \mathbb{E}\Big[\|\mathbf{Y} - \widehat{\mathbf{Y}}\|_F^2\Big],$$

and the rate-distortion function for $(\mathbf{W}, d_{\text{WMSE}})$ equals the standard Gaussian MSE rate-distortion function for $\mathbf{Y}$. In particular, there is a one-to-one correspondence between their rate-distortion curves and optimal codebooks at every distortion level D.

Proof. The proof proceeds in three steps. First, we reduce the WMSE problem for $\mathbf{W}$ to a standard MSE problem for a transformed source by introducing a linear change of variables. Second, we diagonalize the covariance of the transformed source via eigen-decomposition to obtain independent scalar Gaussian components, and apply reverse water-filling to solve the resulting rate-distortion problem. Third, we pull the optimal reconstruction back to $\mathbf{W}$ by inverting the change of variables, treating each structural case of $\boldsymbol{X}$ separately.

Throughout Steps 1 and 2, we work with the vectorized representations

$$\mathbf{w} \;\coloneqq\; \text{vec}(\mathbf{W}) \;\in\; \mathbb{R}^{d_{\text{out}} \cdot d_{\text{in}}}, \qquad \mathbf{y} \;\coloneqq\; \text{vec}(\mathbf{W}\boldsymbol{X}) \;\in\; \mathbb{R}^{d_{\text{out}} \cdot n},$$

which allow us to compute the covariance structure needed for eigen-decomposition. We return to matrix form in Step 3 for the pullback.

Step 1: Reduction to standard MSE. Let $\mathbf{W} \sim \mathcal{N}(0, \sigma_w^2 \boldsymbol{I})$ be the zero-mean i.i.d. Gaussian random matrix and $\boldsymbol{X}$ the deterministic matrix from the theorem statement, with weighted mean squared error distortion (WMSE)

$$d_{\text{WMSE}}(\mathbf{W}, \widehat{\mathbf{W}}) = \mathbb{E}\left[\left\|(\mathbf{W} - \widehat{\mathbf{W}})\boldsymbol{X}\right\|_F^2\right].$$

Introduce the linear change of variables

$$\mathbf{y} \;\coloneqq\; \left(\boldsymbol{X}^\top \otimes \boldsymbol{I}_{d_{\text{out}}}\right)\mathbf{w}, \qquad \widehat{\mathbf{y}} \;\coloneqq\; \left(\boldsymbol{X}^\top \otimes \boldsymbol{I}_{d_{\text{out}}}\right)\widehat{\mathbf{w}},$$

where $\otimes$ is the Kronecker product, so that

$$d_{\text{WMSE}}\Big(\mathbf{W}, \widehat{\mathbf{W}}\Big) = \mathbb{E}\big[\|\mathbf{y} - \widehat{\mathbf{y}}\|_2^2\big].$$

Since $\mathbf{w}$ is zero-mean i.i.d. Gaussian with $\boldsymbol{\Sigma}_{\mathbf{w}} = \sigma_w^2 \boldsymbol{I}$, $\mathbf{y}$ is zero-mean Gaussian with covariance

$$\boldsymbol{\Sigma}_{\mathbf{y}} \;=\; \left(\boldsymbol{X}^\top \otimes \boldsymbol{I}_{d_{\text{out}}}\right)\boldsymbol{\Sigma}_{\mathbf{w}}\left(\boldsymbol{X} \otimes \boldsymbol{I}_{d_{\text{out}}}\right) \;=\; \left(\sigma_w^2\, \boldsymbol{X}^\top \boldsymbol{X}\right) \otimes \boldsymbol{I}_{d_{\text{out}}}.$$

Thus, coding $\mathbf{w}$ under d_{WMSE} at rate R and distortion D is equivalent to coding $\mathbf{y}$ under standard MSE at the same rate and distortion. It, therefore, suffices to solve the standard Gaussian MSE problem for $\mathbf{y}$.

Step 2: Eigen-decomposition and reverse water-filling. However, since $\mathbf{y}$ is a correlated Gaussian vector, we cannot apply water-filling directly. The eigen-decomposition of $\mathbf{\Sigma}_{\mathsf{y}}$ rotates into a basis of independent scalar Gaussians, to which the classical reverse water-filling solution applies. Let

$$\mathbf{\Sigma}_{\mathsf{y}} = \boldsymbol{U}\,\mathrm{diag}(\lambda_1, \lambda_2, \ldots)\,\boldsymbol{U}^\top$$

be the eigen-decomposition of the covariance of $\mathbf{y}$. Expanding in this orthonormal basis,

$$\mathbf{y} = \sum_i \mathsf{v}_i\,\boldsymbol{u}_i, \qquad \widehat{\mathbf{y}} = \sum_i \widehat{\mathsf{v}}_i\,\boldsymbol{u}_i,$$

where the scalar coefficients v_i are independent Gaussians with variances λ_i. The optimal test channel for each scalar component is additive Gaussian with a refinement z_i,

$$\mathsf{v}_i = \widehat{\mathsf{v}}_i + \mathsf{z}_i, \qquad \mathsf{z}_i \sim \mathcal{N}(0, D_i),$$

achieving per-component distortion $D_i \le \lambda_i$.

Since $\boldsymbol{U}$ is orthonormal, the ℓ_2 norm is preserved under the change of basis, and the total MSE distortion of $\mathbf{y}$ decomposes as a sum over components,

$$D = \mathbb{E}\big[\|\mathbf{y} - \widehat{\mathbf{y}}\|_2^2\big] = \sum_i \mathbb{E}\big[(\mathsf{v}_i - \widehat{\mathsf{v}}_i)^2\big] = \sum_i D_i, \qquad D_i \le \lambda_i.$$

and similarly the total rate decomposes as a sum of scalar rates. The rate-distortion function for $\mathbf{y}$ under MSE is therefore

$$R(D) \;=\; \min_{D_i}\; \frac{1}{2}\sum_i \log\!\left(\frac{\lambda_i}{D_i}\right), \qquad \text{subject to } 0 \le D_i \le \lambda_i, \quad \sum_i D_i \le D.$$

This constrained minimization is solved by reverse water-filling, as illustrated in Figure 10. Imagine pouring water up to a level μ over a landscape where each coordinate i has height λ_i. The optimal per-component distortion is $D_i = \min(\mu, \lambda_i)$, where the water level μ is chosen so that $\sum_i D_i = D$. This gives two regimes for each component:

- **Below the water level** ($\lambda_i \le \mu$): the component is fully submerged, so $D_i = \lambda_i$ and no rate is allocated. The source is too weak to be worth encoding.
- **Above the water level** ($\lambda_i > \mu$): the component is only partially submerged, so $D_i = \mu$ and rate is allocated proportional to $\log_2(\lambda_i/\mu)$. Higher-variance components receive more bits.

The total distortion includes contributions from both active and inactive components,

$$D \;=\; \sum_{i\,:\,\lambda_i \le \mu} \lambda_i \;+\; \sum_{i\,:\,\lambda_i > \mu} \mu,$$

while only the active components $\{i : \lambda_i > \mu\}$ contribute to the rate,

$$R(D) \;=\; \frac{1}{2}\sum_{i\,:\,\lambda_i > \mu} \log_2\!\left(\frac{\lambda_i}{\mu}\right).$$

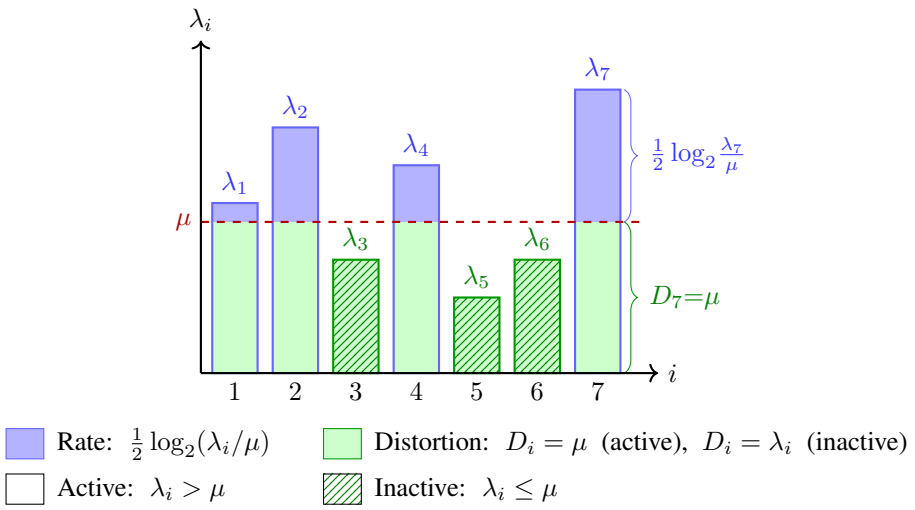


Figure 10: Reverse water-filling method

As the distortion budget D decreases, the water level μ drops, activating more components and allocating rate to progressively weaker directions of $\mathbf{y}$.

Step 3: Pullback to W. Since the change of variables $\mathbf{w} \mapsto \mathbf{y}$ is linear, with an explicit inverse or pseudo-inverse constructed in each of the three cases below, the rate-distortion function derived above for $\mathbf{y}$ under standard MSE is equivalently the rate-distortion function for $\mathbf{w}$ under d_{WMSE}. Since $\mathbf{w} = \mathrm{vec}(\mathbf{W})$ and $\mathbf{y} = \mathrm{vec}(\mathbf{Y})$ are simply vectorized representations of the underlying matrices, we can equivalently write

$$\mathbf{Y} \;=\; \mathbf{W}\boldsymbol{X}, \qquad d_{\mathrm{WMSE}}(\mathbf{W}, \widehat{\mathbf{W}}) \;=\; \mathbb{E}\Big[\big\|(\mathbf{W} - \widehat{\mathbf{W}})\boldsymbol{X}\big\|_F^2\Big] \;=\; \mathbb{E}\Big[\big\|\mathbf{Y} - \widehat{\mathbf{Y}}\big\|_F^2\Big],$$

confirming that the distortion equivalence holds in matrix form as well. The optimal codebook for $\mathbf{W}$ is then recovered from the optimal $\widehat{\mathbf{Y}}$ by inverting the relation $\mathbf{Y} = \mathbf{W}\boldsymbol{X}$, whose precise form depends on the structure of $\boldsymbol{X}$ and is treated separately in each of the three cases below.

***Case 1*: $\boldsymbol{X}$ is square and invertible** ($d_{\mathrm{in}} = n$)**.** Set $\widehat{\mathbf{W}} = \widehat{\mathbf{Y}}\,\boldsymbol{X}^{-1}$. Then, $\widehat{\mathbf{W}}\boldsymbol{X} = \widehat{\mathbf{Y}}$, and therefore

$$\mathbb{E}\Big[\|(\mathbf{W} - \widehat{\mathbf{W}})\boldsymbol{X}\|_F^2\Big] \;=\; \mathbb{E}\Big[\|\mathbf{Y} - \widehat{\mathbf{Y}}\|_F^2\Big].$$

***Case 2*: $\boldsymbol{X}$ is tall and $\boldsymbol{X}^\top\boldsymbol{X}$ is invertible** ($d_{\mathrm{in}} > n$)**.** Let $\boldsymbol{X}^+ = (\boldsymbol{X}^\top\boldsymbol{X})^{-1}\boldsymbol{X}^\top$ denote the left inverse of $\boldsymbol{X}$, which satisfies $\boldsymbol{X}^+\boldsymbol{X} = \boldsymbol{I}_n$. Set $\widehat{\mathbf{W}} = \widehat{\mathbf{Y}}\,\boldsymbol{X}^+$. Then,

$$\widehat{\mathbf{W}}\boldsymbol{X} \;=\; \widehat{\mathbf{Y}}\,(\boldsymbol{X}^+\boldsymbol{X}) \;=\; \widehat{\mathbf{Y}},$$

and therefore

$$\mathbb{E}\Big[\|(\mathbf{W} - \widehat{\mathbf{W}})\boldsymbol{X}\|_F^2\Big] \;=\; \mathbb{E}\Big[\|\mathbf{Y} - \widehat{\mathbf{Y}}\|_F^2\Big].$$

***Case 3*: $\boldsymbol{X}$ is wide and $\boldsymbol{X}\boldsymbol{X}^\top$ invertible** ($d_{\mathrm{in}} < n$)**.** Let $\boldsymbol{X} = \boldsymbol{U}_d\,\boldsymbol{S}_d\,\boldsymbol{V}_d^\top$ be the compact singular value decomposition (SVD) of $\boldsymbol{X}$, where

$$\begin{aligned} &\boldsymbol{U}_d \in \mathbb{R}^{d_{\mathrm{in}} \times d_{\mathrm{in}}} && \text{is orthogonal,} \\ &\boldsymbol{S}_d = \mathrm{diag}(\sigma_1, \ldots, \sigma_{d_{\mathrm{in}}}) && \in \mathbb{R}^{d_{\mathrm{in}} \times d_{\mathrm{in}}}, \\ &\boldsymbol{V}_d \in \mathbb{R}^{n \times d_{\mathrm{in}}}, && \boldsymbol{V}_d^\top\boldsymbol{V}_d = \boldsymbol{I}_{d_{\mathrm{in}}}. \end{aligned}$$

Since $\boldsymbol{X}$ is wide and cannot be inverted directly, we introduce the square, invertible surrogate

$$\widetilde{\boldsymbol{X}} := \boldsymbol{U}_d\,\boldsymbol{S}_d \in \mathbb{R}^{d_{\mathrm{in}} \times d_{\mathrm{in}}}, \qquad \widetilde{\boldsymbol{X}}^{-1} = \boldsymbol{S}_d^{-1}\boldsymbol{U}_d^\top,$$

so that $\boldsymbol{X} = \widetilde{\boldsymbol{X}}\,\boldsymbol{V}_d^\top$. Rather than working with $\mathsf{Y} = \mathsf{W}\boldsymbol{X} \in \mathbb{R}^{d_{\text{out}} \times n}$, which inherits the non-invertibility of $\boldsymbol{X}$, we instead work with the compressed surrogate

$$\widetilde{\mathsf{Y}} \;\coloneqq\; \mathsf{W}\,\widetilde{\boldsymbol{X}} \;\in\; \mathbb{R}^{d_{\text{out}} \times d_{\text{in}}}.$$

Using $\boldsymbol{X} = \widetilde{\boldsymbol{X}}\boldsymbol{V}_d^\top$, the original outputs are recovered as

$$\mathsf{Y} = \mathsf{W}\boldsymbol{X} = \mathsf{W}\,\widetilde{\boldsymbol{X}}\,\boldsymbol{V}_d^\top = \widetilde{\mathsf{Y}}\,\boldsymbol{V}_d^\top.$$

and since $\boldsymbol{V}_d$ has orthonormal columns, right multiplication by $\boldsymbol{V}_d^\top$ preserves the Frobenius norm, giving

$$\mathbb{E}\Big[\|\mathsf{Y} - \widehat{\mathsf{Y}}\|_F^2\Big] \;=\; \mathbb{E}\Big[\|\widetilde{\mathsf{Y}} - \widehat{\widetilde{\mathsf{Y}}}\|_F^2\Big].$$

It therefore suffices to solve the rate-distortion problem for $\widetilde{\mathsf{Y}}$. Since $\widetilde{\boldsymbol{X}}$ is square and invertible, this reduces to Case 1, and the same reverse water-filling applied to $\boldsymbol{\Sigma}_{\mathsf{y}}$ yields the optimal reconstruction $\widehat{\widetilde{\mathsf{Y}}}$. The reconstruction $\widehat{\mathsf{W}}$ is then recovered via

$$\widehat{\mathsf{W}} \;=\; \widehat{\widetilde{\mathsf{Y}}}\,\widetilde{\boldsymbol{X}}^{-1}.$$

To verify that this achieves the correct distortion on the original samples, note that the forward map gives $\widehat{\mathsf{W}}\boldsymbol{X} = \widehat{\widetilde{\mathsf{Y}}}\,\boldsymbol{V}_d^\top$. Thus, the output error satisfies $(\mathsf{W} - \widehat{\mathsf{W}})\boldsymbol{X} = (\widetilde{\mathsf{Y}} - \widehat{\widetilde{\mathsf{Y}}})\,\boldsymbol{V}_d^\top$. Using the norm equivalence established above, we conclude

$$\mathbb{E}\Big[\|(\mathsf{W} - \widehat{\mathsf{W}})\boldsymbol{X}\|_F^2\Big] \;=\; \mathbb{E}\Big[\|\widetilde{\mathsf{Y}} - \widehat{\widetilde{\mathsf{Y}}}\|_F^2\Big] \;=\; \mathbb{E}\Big[\|\mathsf{Y} - \widehat{\mathsf{Y}}\|_F^2\Big].$$

In all three cases, the recovery $\widehat{\mathsf{W}}$ is constructed so that

$$\mathbb{E}\Big[\|(\mathsf{W} - \widehat{\mathsf{W}})\boldsymbol{X}\|_F^2\Big] \;=\; \mathbb{E}\Big[\|\mathsf{Y} - \widehat{\mathsf{Y}}\|_F^2\Big],$$

confirming that the WMSE distortion on W equals the standard MSE distortion on Y in all cases.

Hence, this theorem shows:

1. **Distortion equivalence.** For any reconstruction $\widehat{\mathsf{W}}$, setting $\widehat{\mathsf{Y}} = \widehat{\mathsf{W}}\boldsymbol{X}$ gives
$$d_{\text{WMSE}}(\mathsf{W}, \widehat{\mathsf{W}}) \;=\; \mathbb{E}\Big[\big\|\mathsf{Y} - \widehat{\mathsf{Y}}\big\|_F^2\Big].$$
2. **Rate-distortion equivalence.** The rate-distortion function for $(\mathsf{W}, d_{\text{WMSE}})$ equals the standard Gaussian MSE rate-distortion function for Y, given by reverse water-filling over the eigenvalues $\{\lambda_i\}$ of $\boldsymbol{\Sigma}_{\mathsf{y}}$:
$$R(D) \;=\; \frac{1}{2}\sum_{i\,:\,\lambda_i > \mu} \log_2\!\left(\frac{\lambda_i}{\mu}\right),$$
where μ is chosen so that $\sum_i \min(\mu, \lambda_i) = D$.
3. **Codebook recovery.** The optimal reconstruction $\widehat{\mathsf{W}}$ is recovered from the optimal $\widehat{\mathsf{Y}}$ by inverting $\mathsf{Y} = \mathsf{W}\boldsymbol{X}$. The precise form depends on the structure of $\boldsymbol{X}$:
   - **Square** ($d_{\text{in}} = n$, $\boldsymbol{X}$ invertible): $\widehat{\mathsf{W}} = \widehat{\mathsf{Y}}\,\boldsymbol{X}^{-1}$.
   - **Tall** ($d_{\text{in}} > n$), $\boldsymbol{X}^\top\boldsymbol{X}$ invertible): $\widehat{\mathsf{W}} = \widehat{\mathsf{Y}}\,(\boldsymbol{X}^\top\boldsymbol{X})^{-1}\boldsymbol{X}^\top$.
   - **Wide** ($d_{\text{in}} < n$, via compact SVD $\boldsymbol{X} = \boldsymbol{U}_d\boldsymbol{S}_d\boldsymbol{V}_d^\top$): $\widehat{\mathsf{W}} = \widehat{\widetilde{\mathsf{Y}}}\,\widetilde{\boldsymbol{X}}^{-1}$, where $\widetilde{\boldsymbol{X}} \coloneqq \boldsymbol{U}_d\boldsymbol{S}_d$.

Moreover, in Case 2, where $\mathsf{w} \mapsto \mathsf{y}$ is many-to-one, the optimal test channel allocates zero bits to the null space of this map, so the non-invertibility does not affect d_{WMSE}. In all three cases, the recovery map from $\widehat{\mathsf{Y}}$ to $\widehat{\mathsf{W}}$ is linear, determined by $\boldsymbol{X}$ alone, and preserves the distortion. We refer to this as the *pullback*.

□

E.2 Successive Refinability of the Transformed Source

Theorem 4 established that the rate-distortion function for $\mathbf{W}$ under d_{WMSE} coincides with the standard MSE rate-distortion function for the transformed source $\mathbf{y}$ at any single distortion level. We now show that $\mathbf{y}$ is itself successively refinable under standard MSE.

Theorem 5 (Correlated Gaussian under MSE is successively refinable)**.** *Let $\mathbf{y}$ be zero-mean Gaussian with covariance $\mathbf{\Sigma}_{\boldsymbol{y}} = \boldsymbol{U}\,\mathrm{diag}(\lambda_1, \lambda_2, \dots)\boldsymbol{U}^\top$, as established in the proof of Theorem 4. Then $\mathbf{y}$ is successively refinable under the mean squared error distortion measure. That is, for any nested distortion levels $D_1 > D_2 > \cdots > D_K > 0$, there exists a single embedded bitstream whose k-th prefix achieves the optimal rate-distortion pair $(R(D_k), D_k)$ simultaneously for all k, at the same total rate as K independent codes each optimally designed for its own distortion level. Moreover, the code is realized on blocks of $\mathbf{W}$, hence, its stage-k reconstructions satisfy the same guarantees under d_{WMSE}.*

Proof. We treat each realization of the full matrix $\mathbf{W} \in \mathcal{W} \triangleq \mathbb{R}^{d_{\mathrm{out}} \times d_{\mathrm{in}}}$ as an individual source symbol. A block of length N is an i.i.d. sequence $\mathbf{W}^N = (\mathbf{W}_1, \dots, \mathbf{W}_N)$, where $\boldsymbol{X}$ remains fixed across all N realizations, and we consider $N \to \infty$. Via the transformation in Theorem 4, each matrix $\mathbf{W}$ is mapped to a vector $\mathbf{v} \in \mathbb{R}^m$ with $m = d_{\mathrm{out}} \min(d_{\mathrm{in}}, n)$. Consequently, the matrix source is an instance of the vector formulation in Definition 2 with symbol alphabet $\mathcal{W}$. The codebooks act directly on the eigen-basis coordinates v_i, rather than on the original entries of $\mathbf{W}$.

By Theorem 4, $\mathbf{y}$ is zero-mean Gaussian with covariance $\mathbf{\Sigma}_{\mathsf{y}} = (\sigma_w^2 \boldsymbol{X}^\top \boldsymbol{X}) \otimes \boldsymbol{I}_{d_{\mathrm{out}}}$. Let

$$\mathbf{\Sigma}_{\mathsf{y}} \;=\; \boldsymbol{U}\,\mathrm{diag}(\lambda_1, \lambda_2, \dots)\,\boldsymbol{U}^\top$$

be its eigen-decomposition. Transforming to the eigen-basis via $\mathbf{v} = \boldsymbol{U}^\top \mathbf{y}$ yields independent scalar Gaussian components $\mathsf{v}_i \sim \mathcal{N}(0, \lambda_i)$. Since the components are independent, it suffices to show successive refinability for each scalar v_i separately, then sum over components.

Scalar test channel construction. For each component v_i, superscripts denote the refinement stage, while subscripts index the source component. For example, $\widehat{\mathsf{v}}_i^{(k)}$ is the reconstruction of component i after stage k, and $\mathsf{z}_i^{(k)}$ is the Gaussian increment added at stage k. Define nested distortion levels $D_i^{(1)} > D_i^{(2)} > 0$. Then, define mutually independent Gaussian increments

$$\mathsf{z}_i^{(1)} \sim \mathcal{N}\Big(0,\, \lambda_i - D_i^{(1)}\Big), \quad \mathsf{z}_i^{(2)} \sim \mathcal{N}\Big(0,\, D_i^{(1)} - D_i^{(2)}\Big), \quad \mathsf{z}_i \sim \mathcal{N}\Big(0,\, D_i^{(2)}\Big),$$

and construct successive reconstructions

$$\widehat{\mathsf{v}}_i^{(0)} \;=\; 0, \qquad \widehat{\mathsf{v}}_i^{(1)} \;=\; \widehat{\mathsf{v}}_i^{(0)} + \mathsf{z}_i^{(1)}, \qquad \widehat{\mathsf{v}}_i^{(2)} \;=\; \widehat{\mathsf{v}}_i^{(1)} + \mathsf{z}_i^{(2)}, \qquad \mathsf{v}_i \;=\; \widehat{\mathsf{v}}_i^{(2)} + \mathsf{z}_i.$$

Each stage k refines the previous estimate by adding an independent Gaussian increment of variance $D_i^{(k-1)} - D_i^{(k)}$. By construction, $\mathbb{E}\big[(\mathsf{v}_i - \widehat{\mathsf{v}}_i^{(k)})^2\big] = D_i^{(k)}$ for $k = 1, 2$, and since the increments are mutually independent the Markov chain holds by construction,

$$\widehat{\mathsf{v}}_i^{(1)} \;\to\; \widehat{\mathsf{v}}_i^{(2)} \;\to\; \mathsf{v}_i$$

No rate penalty. The rate $\Delta R_i^{(1)}$ is the rate needed to encode $\mathsf{z}_i^{(1)}$, i.e. the information required to describe the first-stage reconstruction $\widehat{\mathsf{v}}_i^{(1)}$ from scratch. The rate $\Delta R_i^{(2)}$ is the additional rate needed to encode $\mathsf{z}_i^{(2)}$, i.e. the new information required to describe how the reconstruction improves from $\widehat{\mathsf{v}}_i^{(1)}$ to $\widehat{\mathsf{v}}_i^{(2)}$. Formally,

$$\Delta R_i^{(1)} \;=\; I\Big(\mathsf{v}_i;\, \widehat{\mathsf{v}}_i^{(1)}\Big) \;=\; \frac{1}{2}\log\frac{\lambda_i}{D_i^{(1)}},$$

$$\Delta R_i^{(2)} \;=\; I\Big(\mathsf{v}_i;\, \widehat{\mathsf{v}}_i^{(2)} \,\Big|\, \widehat{\mathsf{v}}_i^{(1)}\Big) \;=\; h\Big(\mathsf{v}_i \mid \widehat{\mathsf{v}}_i^{(1)}\Big) - h\Big(\mathsf{v}_i \mid \widehat{\mathsf{v}}_i^{(1)}, \widehat{\mathsf{v}}_i^{(2)}\Big) \;=\; \frac{1}{2}\log\frac{D_i^{(1)}}{D_i^{(2)}}.$$

The total two-stage rate is equal to the one-shot rate at $D_i^{(2)}$:

$$\Delta R_i^{(1)} + \Delta R_i^{(2)} \;=\; \frac{1}{2}\log\frac{\lambda_i}{D_i^{(1)}} + \frac{1}{2}\log\frac{D_i^{(1)}}{D_i^{(2)}} \;=\; \frac{1}{2}\log\frac{\lambda_i}{D_i^{(2)}} \;=\; R_i(D_i^{(2)}),$$

confirming no rate penalty for the i-th component. The Markov consistency condition

$$I\left(\mathsf{v}_i;\, \widehat{\mathsf{v}}_i^{(1)} \,\middle|\, \widehat{\mathsf{v}}_i^{(2)}\right) = 0$$

holds because $\mathsf{v}_i = \widehat{\mathsf{v}}_i^{(2)} + \mathsf{z}_i$, where the final increment z_i is mutually independent of the earlier reconstructions $\widehat{\mathsf{v}}_i^{(1)}$ and $\widehat{\mathsf{v}}_i^{(2)}$ by construction. Hence, conditioned on $\widehat{\mathsf{v}}_i^{(2)}$, the residual $\mathsf{v}_i - \widehat{\mathsf{v}}_i^{(2)} = \mathsf{z}_i$ carries no information about $\widehat{\mathsf{v}}_i^{(1)}$, so v_i is conditionally independent of $\widehat{\mathsf{v}}_i^{(1)}$ given $\widehat{\mathsf{v}}_i^{(2)}$, establishing the Markov chain $\widehat{\mathsf{v}}_i^{(1)} \to \widehat{\mathsf{v}}_i^{(2)} \to \mathsf{v}_i$.

The same construction extends to K stages by defining K independent increments of variances $\lambda_i - D_i^{(1)}$, $D_i^{(1)} - D_i^{(2)}$, $\ldots$, $D_i^{(K-1)} - D_i^{(K)}$, and the joint Markov chain

$$\widehat{\mathbf{y}}^{(1)} \;\to\; \widehat{\mathbf{y}}^{(2)} \;\to\; \cdots \;\to\; \widehat{\mathbf{y}}^{(K)} \;\to\; \mathbf{y}$$

holds since the scalar Markov chains hold independently for each component, with each stage being a strictly finer approximation of $\mathbf{y}$ than the previous. Moreover, Figures 11 and 12 illustrate the three possible activation patterns: at the first stage, case (a) starts with only one component active while the other remains inactive, case (b) starts with both components already active, and case (c) starts with both components inactive. In all cases the ΔR brackets confirm that the incremental rate at each stage equals exactly $R(D_{k+1}) - R(D_k)$. We next demonstrate the operational achievability of these test channels.

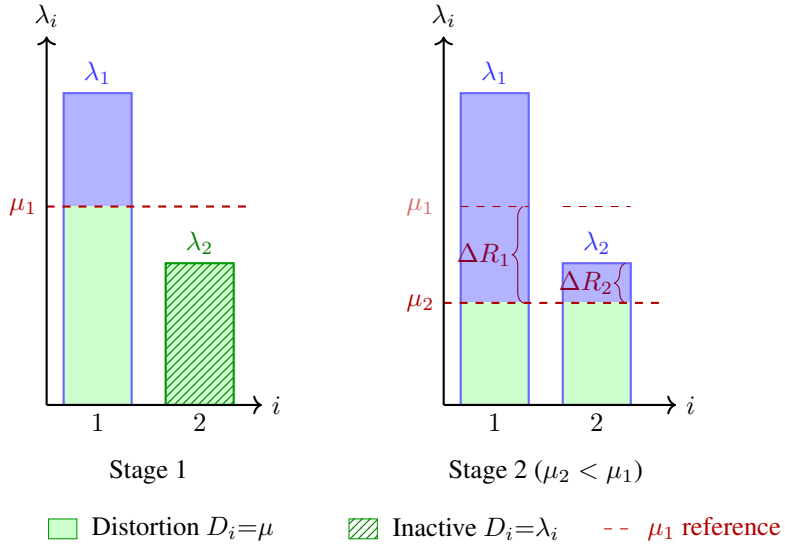


Figure 11: Successive refinement under reverse water-filling: case (a) shows at Stage 1, component 1 is active ($\lambda_1 > \mu_1$) while component 2 is inactive ($\lambda_2 \le \mu_1$). At Stage 2, the water level drops to $\mu_2 < \mu_1$, activating component 2 and refining both.

Under reverse water-filling, the per-component distortion levels $D_i^{(k)}$ that achieve the rate-distortion function at overall distortion D_k are given explicitly by

$$D_i^{(k)} \;=\; \min(\lambda_i,\, \mu_k)\,,$$

where $\mu_k > 0$ is the water level chosen so that $\sum_i D_i^{(k)} = D_k$. Nested distortion levels $D_1 > D_2 > \cdots > D_K$ therefore correspond to a strictly decreasing sequence of water levels $\mu_1 > \mu_2 > \cdots > \mu_K$.

As μ decreases across refinement stages, the distortion assigned to each active coordinate i (where $\lambda_i > \mu$) equals μ and therefore decreases monotonically, while inactive coordinates (where $\lambda_i \leq \mu$) retain distortion $D_i = \lambda_i$ and contribute zero rate at that stage, remaining unchanged until μ drops below λ_i and activates them. Thus, lowering the water level strictly refines the reconstruction: each stage produces a finer estimate of $\mathbf{y}$, and the test channels nest properly across all K distortion levels.

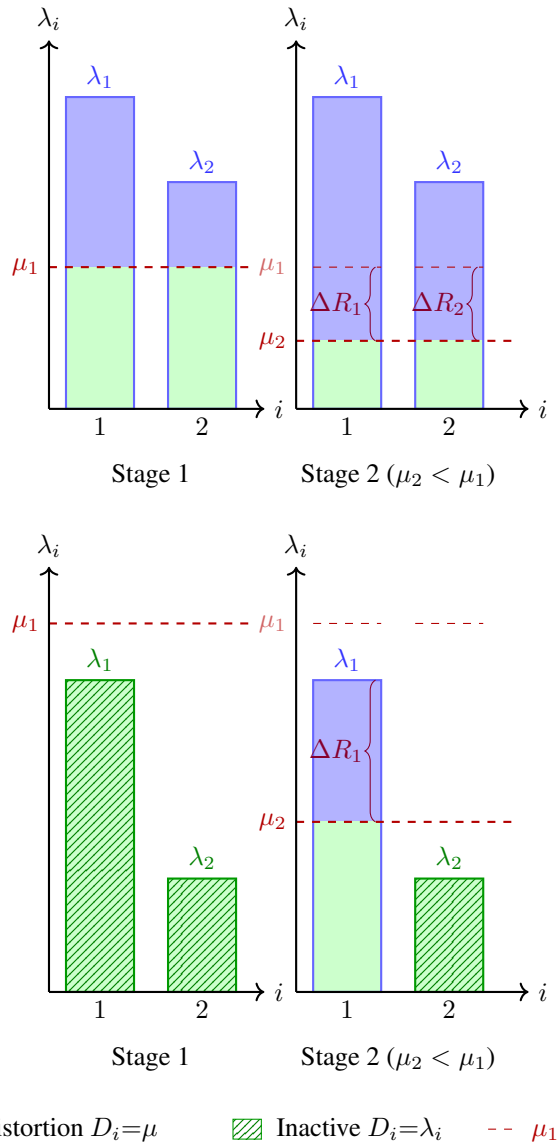


Figure 12: Successive refinement under reverse water-filling: case (b) shows both components active from Stage 1 ($\lambda_1, \lambda_2 > \mu_1$); case (c) shows both inactive at Stage 1 ($\lambda_1, \lambda_2 \leq \mu_1$), with component 1 activating only at Stage 2 when μ_2 drops below λ_1. In all cases, the incremental rates ΔR sum to the single-shot rate at the finest distortion level.

Achievability. Applying the change of variables of Theorem 4 symbol by symbol gives $\mathbf{v}_t \in \mathbb{R}^m$ for $t = 1, \ldots, N$. Since the map is deterministic and applied identically at every t, the sequence $\mathbf{v}^N$ is i.i.d. with components $\mathsf{v}_{t,i} \sim \mathcal{N}(0, \lambda_i)$ independent across i, and m is fixed by the layer dimensions rather than growing with N.

Generation of codebooks. Apply the codebook construction of Theorem 3 to $\mathbf{v}^N$, with λ_i in place of σ_i^2 and $D_i^{(k)} = \min(\lambda_i, \mu_k)$. For each stage k and each active component i, this generates

$2^{N(R_{k,i}-R_{k-1,i})}$ increment sequences of length N, drawn i.i.d. $\mathcal{N}\Big(0,\, D_i^{(k-1)} - D_i^{(k)}\Big)$. Inactive components receive no codebook and no rate at that stage. Reveal these codebooks to the encoder and the decoder.

Encoding. The stage-k encoder $i_N^{(k)}$ transforms each symbol of $\mathbf{W}^N$, then performs the stage-k encoding of Theorem 3 on $\mathbf{v}^N$: for each active component i it searches independently for an index whose increment sequence is jointly distortion typical with the residual of component i. If there is more than one such index, send the least. If there is no such index, default to the first index. The transmitted stage-k index is the concatenation of these parallel indices, at incremental rate $\Delta R^{(k)}$.

Decoding. The stage-k decoder $g_N^{(k)}$ separates the received indices by component, sums the selected increments through stage k to obtain $\widehat{\mathbf{v}}^{(k)N}$, and applies the pullback of Theorem 4 symbol by symbol to recover $\widehat{\mathbf{W}}_t^{(k)}$ for $t = 1, \ldots, N$. A stage-k codeword is therefore the length-N sequence $\widehat{\mathbf{W}}^{(k)N} = (\widehat{\mathbf{W}}_1^{(k)}, \ldots, \widehat{\mathbf{W}}_N^{(k)})$, each letter of which is a $d_{\text{out}} \times d_{\text{in}}$ matrix. The change of variables and the pullback are deterministic and known to encoder and decoder.

Calculation. The pullback preserves the distortion, so the distortion typical sets of the two problems coincide and typicality is applied to the unweighted MSE on $\mathbf{v}$ alone. The error analysis of Theorem 3 therefore transfers unchanged: each component fails with probability $P_{e,i}^{(k)} \le \exp\big(-2^{N(\varepsilon_{k,i}-\delta_{k,i})}\big)$, and since m is finite and independent of N, the union bound gives $P_{e,\text{total}}^{(k)} \to 0$. On the event that all parallel codebooks succeed,

$$\frac{1}{N}\sum_{t=1}^{N}\mathbb{E}\left[\left\|\left(\mathbf{W}_t - \widehat{\mathbf{W}}_t^{(k)}\right)\boldsymbol{X}\right\|_F^2\right] \;\longrightarrow\; \sum_{i=1}^{m} D_i^{(k)} \;=\; D^{(k)}, \qquad k = 1, \ldots, K.$$

The encoders $i_N^{(k)}$ and decoders $g_N^{(k)}$ thus satisfy Definition 2 at cumulative rates $R^{(k)}$. Hence $\mathbf{y}$ is successively refinable under MSE, and the code just constructed achieves the same rates and distortions on blocks of $\mathbf{W}$ under d_{WMSE}.

□

F FULL ALGORITHM

Algorithm 2 presents the complete Drop-by-Drop training procedure, expanding the condensed version given as Algorithm 1 in the main text.

Beam search updates assignments b using only the final-stage reconstruction error $L^{(M)}$, not the full composite loss $\mathcal{L}_{\text{DbyD}}$; evaluating all M partial reconstructions per candidate would multiply beam search cost by M. Codebook values C are optimized against the full weighted objective. Block-level fine-tuning optimizes θ against the fully-reconstructed ($k = M$) output only; this is an intentional design choice made for the same complexity reason as beam search above. We do not apply the Matryoshka weighting during fine-tuning, and rely on the codebook-level loss to preserve the nested structure. Both these decisions are supported empirically by the graceful degradation in perplexity and zero-shot accuracy across $k \leq M$ shown in Section 6.

Following AQLM, both phases use early stopping. For codebook optimization, we use a relative-improvement tolerance $\tau = 0.01$, checked once per epoch. For fine-tuning, we use early-stopping after 3 consecutive non-improving epochs on validation loss.

The block size B_H is selected per layer from the candidate set $\{256, 128, 64, 32\}$, taking the largest value that evenly divides d_{in}; if none does, $B_H = 1$ (no blocking is applied).

Algorithm 2 Drop-by-Drop Quantization with Fast Walsh-Hadamard Transformation (FWHT)

Require: model, data

X_{block} := model.input_embeddings(data)
for $i = 1, \ldots,$ `model.num_layers` **do**
  block := model.get_block(i)
  Y_{block} := block(X_{block})
  for layer ∈ linear_layers(block) **do**
    W := layer.weight, d_{in} := columns(W)
    $XX^\top$:= accumulate_hessian(layer, X_{block})
    Select block size B_H and sample random signs $S \sim \mathcal{U}(\{-1, 1\})^{d_{\text{in}}}$
    $W \leftarrow \text{BlockFWHT}_{\text{1D}}(W \odot S, B_H)$
    $XX^\top \leftarrow \text{BlockFWHT}_{\text{2D}}(XX^\top \odot (SS^\top), B_H)$
    C, b, s := k_means_initialize(W)
    while loss improves by at least τ **do**
      for $k = 1, \ldots, M$ **do**
        $\widehat{W}^{(k)} \leftarrow$ partial reconstruction using the first k codebooks
        $L^{(k)} \leftarrow \frac{1}{d_{\text{out}}} \text{tr}\left[(W - \widehat{W}^{(k)}) \left(\frac{1}{n} XX^\top\right) (W - \widehat{W}^{(k)})^\top\right]$
      end for
      Total loss: $\mathcal{L}_{\text{DbyD}} = \sum_k \alpha_k L^{(k)}$
      Update codebooks C by gradient descent on $\mathcal{L}_{\text{DbyD}}$
      Update assignments b by beam search against $L^{(M)}$ only
    end while
    layer.weight := AQLMFormat(C, b, s)
    layer.rotation_signs := S, layer.block_size := B_H
  end for
  θ := trainable_parameters(block)
  while loss improves by at least τ **do**
    $\mathcal{L}_{\text{FT}} := \|\texttt{block}(X_{\text{block}}) - Y_{\text{block}}\|_2^2$ ▷ targets M; not re-weighted per prefix
    θ := adam(θ, $\partial \mathcal{L}_{\text{FT}} / \partial \theta$)
  end while
  X_{block} := block(X_{block})
end for

Algorithms 3 and 4 give the corresponding inference-time reconstruction procedures for the standard DbyD and Hadamard-rotated DbyD variant, respectively; the latter rotates the input activation rather than the dequantized weight. The rotation is invertible; rather than undoing it once and materializing a dense weight, we apply the identical rotation to the layer's input activation, x, on every forward

pass, while the weight remains permanently stored in its rotated, compressed form. This introduces no additional VRAM overhead relative to vanilla AQLM: the codebooks and codes never leave their compressed representation, and the weight is reconstructed only transiently for each matmul. The added cost is a single block-Hadamard transform over the input activation vector and not the weight matrix. Hence, it remains negligible relative to the matmul calculation.

Algorithm 3 Drop-by-Drop Weight Reconstruction for Resource-Constrained Inference

Require: Compressed weight components: `codes`, `codebooks`, `scales`
Require: Desired number of codebooks k, total codebooks M
Require: Layer meta-parameters: rotation signs S, block size B_H
if k is specified **then**
  $k \leftarrow \min(k, M)$
else
  $k \leftarrow M$
end if
$W \leftarrow$ `Dequantize`(codes, codebooks, scales, k)
return W

Algorithm 4 Drop-by-Drop Forward Pass for Resource-Constrained Inference for FWHT

Require: Compressed weight components: `codes`, `codebooks`, `scales`
Require: Input activation x
Require: Desired number of codebooks k, total codebooks M
Require: Layer meta-parameters: rotation signs S, block size B_H
if k is specified **then**
  $k \leftarrow \min(k, M)$
else
  $k \leftarrow M$
end if
if S is present and $B_H > 1$ **then**
  $x \leftarrow x \odot S$ ▷ Apply sign flip
  $x \leftarrow \text{BlockFWHT}_{1\text{D}}(x, B_H)$ ▷ Apply rotation
end if
$\widehat{W}^{(k)} \leftarrow$ `Dequantize`(codes, codebooks, scales, k)
return $\widehat{W}^{(k)} x$

G Ablation Study

G.1 Tradeoff Between Codebook Precision and Codebook Count

A key design question in multi-codebook quantization is whether performance is better preserved by using many low-precision codebooks or fewer high-precision ones, under a fixed overall bit budget. Prior observations from AQLM suggest that increasing codebook precision can be more beneficial than increasing the number of codebooks when the average bit rate is held constant.

To validate whether this behavior persists under our Drop-by-Drop training strategy, we conduct a controlled experiment on Gemma-2B Team et al. (2024). We evaluate perplexity on WikiText2 and C4 under an *extreme* quantization regime, where the average effective bit rate is approximately 2 bits. We compare two configurations trained with identical optimization settings and Drop-by-Drop loss, differing only in how the bit budget is allocated across codebooks. Specifically, we contrast a configuration with fewer high-precision codebooks against one with more low-precision codebooks. Despite having comparable average bit rates (e.g., average bit ≈ 2 bits), the high-precision configuration consistently achieves substantially lower perplexity on both datasets.

Table 4: Perplexity on WikiText2 and C4 (↓) for codebook precision vs. codebook count ablation.

Configuration	Bits per Codebook	# Codebooks	WikiText2	C4
High precision	8	2	**19.22**	**23.37**
More codebooks	4	4	34.02	77.17

These results indicate that, under tight bit budgets, allocating more bits per codebook is significantly more effective than increasing the number of codebooks. This finding supports the core design of Drop-by-Drop, which prioritizes strong intermediate reconstructions and enables robust performance even when only a small subset of high-precision codebooks is active. This property is critical for flexible deployment across devices with heterogeneous resource constraints.

G.2 Matryoshka Loss Coefficients

We next study the impact of different Matryoshka loss weightings on performance. The Matryoshka loss supervises multiple intermediate reconstructions corresponding to different subsets of active codebooks, and its weighting scheme directly influences which intermediate representations are emphasized during training.

We experiment with a wide range of weighting strategies, including highly skewed coefficients, uniform averages, and exponentially increasing and decreasing coefficients, as well as variants that selectively supervise specific intermediate codebook combinations, such as:

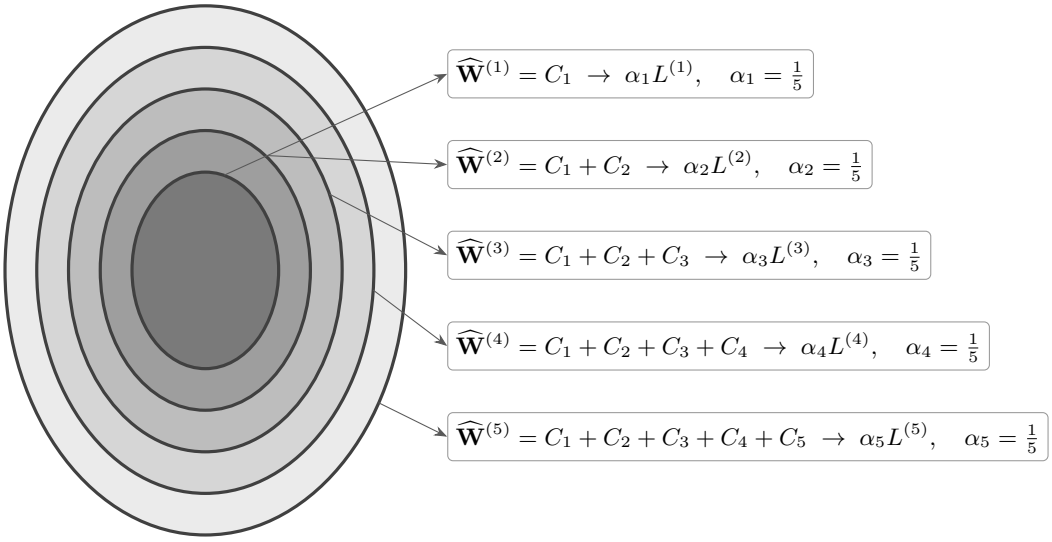
- **3W**: $\alpha_3 = 1$, all other α_k values set to zero.
- **345W**: $\alpha_3 = \alpha_4 = \alpha_5 = \frac{1}{3}$, all other α values set to zero.
- **35W**: $\alpha_3 = \alpha_5 = \frac{1}{2}$, all other α values set to zero.

Overall, we find that aggressively weighting early or late reconstructions alone does not consistently improve performance. Instead, selectively supervising intermediate reconstructions yields the most stable behavior under Drop-by-Drop training. Two visual examples are provided in Figure 13.

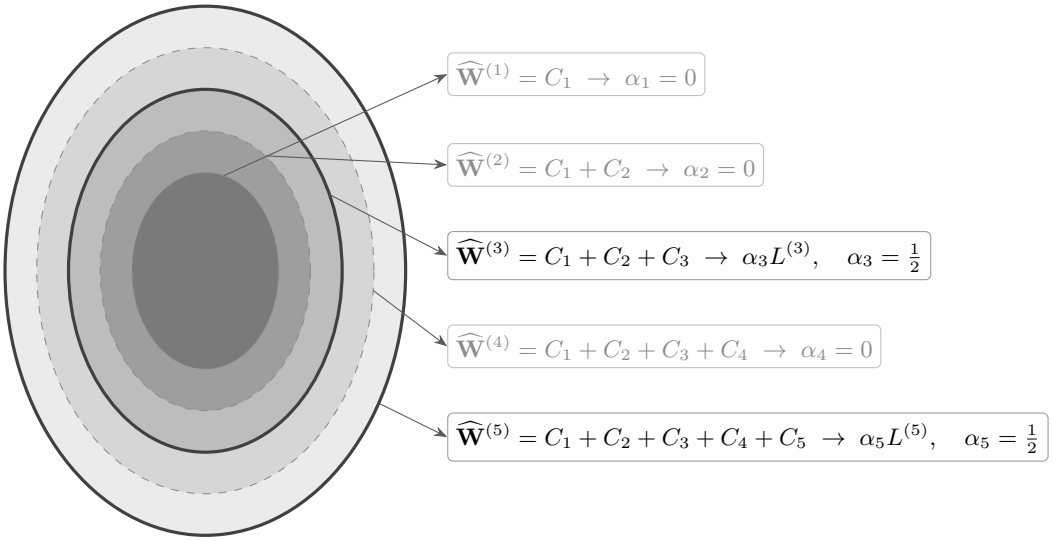
This observation aligns with our goal of maintaining strong performance at intermediate bit rates, rather than optimizing exclusively for the highest-fidelity reconstruction.

G.3 Tradeoff Between Number of Supported Bitwidths and Performance

We further investigate whether restricting the set of supported bitwidths (and concentrating supervision on fewer intermediate reconstructions) yields better performance at those bitwidths. To investigate this, we compare two loss variants specialized for bitwidths $\{3, 4\}$ against the three-bitwidth baseline $\{3, 4, 5\}$. Hence, for this ablation, models are trained with a maximum of $M = 4$ codebooks (rather



(a) DbyD (Uni.): every prefix is supervised.



(b) DbyD (35W): only $k = 3$ and $k = 5$ are supervised.

Figure 13: Matryoshka coefficient variants, with $\mathcal{L}_{\text{DbyD}} = \sum_k \alpha_k L^{(k)}$ and $M = 5$. Each ellipse is the partial reconstruction $\widehat{\mathbf{W}}^{(k)}$ formed from the first k codebooks, where C_m denotes the contribution of the m-th codebook, so nesting encodes prefix containment. Shading intensity indicates distortion, with the coarsest reconstruction in darkest gray; sizes and shades are illustrative and carry no quantitative meaning.

than the $M = 5$ used in Section 6), so AQLM (Ind.) and AQLM (Drop) coincide exactly at the 4-bit setting, as neither involves dropping at that precision.

- **34W**: $\alpha_1 = \alpha_2 = 0; \alpha_3 = \alpha_4 = \frac{1}{2}$.
- **234W**: $\alpha_1 = 0; \alpha_2 = \alpha_3 = \alpha_4 = \frac{1}{3}$.

As shown in Tables 5 and 6, both 34W and 234W achieve lower perplexity at 3 and 4 bits compared to the full three-bitwidth DbyD model, [1] confirming that narrowing the supervision target improves performance at the supported bitwidths. However, this comes at the cost of flexibility: a model trained with 34W does not support a 5-bit precision without retraining. This tradeoff between adaptability and peak performance is a key consideration when selecting a loss configuration for deployment.

Table 5: Perplexity on WIKITEXT2 (↓) for supported bitwidths vs performance ablation.

Model	Bits	AQLM (Ind.)	AQLM (Drop)	DbyD (34W)	DbyD (234W)
Qwen 0.5B	4	13.76	13.76	**13.91**	13.93
	3	15.43	26.81	**23.71**	24.07
Qwen 3B	4	7.71	7.71	**7.74**	7.77
	3	8.25	39.83	**9.39**	9.47
Qwen 7B	4	7.08	7.08	**7.13**	7.16
	3	7.47	9.01	**8.51**	8.64

Table 6: Perplexity on C4 (↓) for supported bitwidths vs performance ablation.

Model	Bits	AQLM (Ind.)	AQLM (Drop)	DbyD (34W)	DbyD (234W)
Qwen 0.5B	4	18.73	18.73	**18.90**	19.11
	3	21.87	33.46	32.40	**30.72**
Qwen 3B	4	11.72	11.72	**11.76**	11.84
	3	12.94	48.40	**13.91**	14.12
Qwen 7B	4	10.94	10.94	**10.92**	11.04
	3	11.85	13.41	**12.75**	12.97

G.4 COMPUTATIONAL AND STORAGE BENEFITS

Drop-by-Drop demonstrates that a single quantized model can serve multiple resource budgets without retraining or maintaining separate checkpoints. Conventional methods such as AQLM require a separate model per target configuration, which in our setup means three independently trained and stored models at 3×8, 4×8, and 5×8. As shown in Figure 14, DbyD trains and stores only once showcasing significant training hours and disk space benefits.

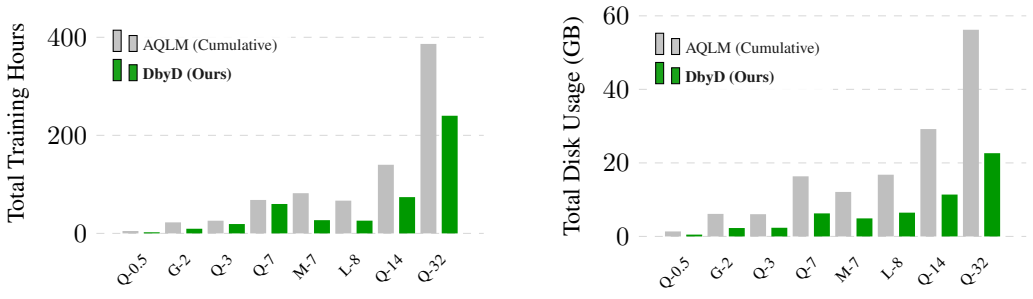


Figure 14: Training time and disk space required to produce a suite of quantized models

[1] See the DbyD (Uni.) and DbyD (35W) rows in Tables 7 and 8 for these values.

H Quantitative Empirical Results

All perplexity and accuracy results report the mean over *three* independent runs. For perplexity, the $\pm$ value is the standard deviation across runs. For zero-shot accuracy, the $\pm$ value is the root mean square (RMS) of the standard errors reported by the evaluation code for each run,

$$\mathrm{SE}_{\mathrm{RMS}} = \sqrt{\frac{1}{N}\sum_{i=1}^{N}\mathrm{SE}_i^2},$$

where N is the number of runs and SE_i is the standard error of run i. Tables 7, 8, and 9 report the values plotted in Figures 5, 15, and 6. Lower perplexity (↓) and higher accuracy (↑) indicate better performance.

H.1 Perplexity and Average Zero-Shot Accuracy Values

Table 7: Perplexity on C4 (↓). Values plotted in Figure 5.

Model	Code-books	AQLM (Ind.)	AQLM (Drop)	DbyD (Uni.)	DbyD (35W)	DbyD (Had.+35W)
Gemma 2B†	5×8	12.50	12.50	12.47	12.06 ± 0.18	11.79 ± 0.00
	4×8	12.91	13.28	14.02	13.59 ± 0.44	13.23 ± 0.37
	3×8	15.13	18.37	18.26	15.93 ± 0.53	17.07 ± 1.15
Mistral 7B	5×8	7.93 ± 0.62	7.93 ± 0.62	7.70 ± 0.04	7.58 ± 0.01	7.54 ± 0.00
	4×8	7.73 ± 0.01	8.74 ± 0.39	8.31 ± 0.05	8.08 ± 0.07	7.71 ± 0.00
	3×8	8.23 ± 0.01	104.10 ± 77.90	16.09 ± 9.05	9.14 ± 0.18	8.32 ± 0.00
MetaLlama 8B	5×8	8.71 ± 0.01	8.71 ± 0.01	9.01 ± 0.01	8.78 ± 0.00	8.63 ± 0.00
	4×8	9.29 ± 0.01	28.22 ± 27.49	10.66 ± 0.36	10.39 ± 0.21	9.38 ± 0.04
	3×8	10.73 ± 0.01	$7.8\times10^4\pm1.3\times10^5$	20.61 ± 2.93	12.97 ± 0.36	11.53 ± 0.05
Qwen 0.5B	5×8	17.87 ± 0.02	17.87 ± 0.02	18.30 ± 0.02	18.00 ± 0.03	17.87 ± 0.00
	4×8	18.73 ± 0.02	20.91 ± 0.24	21.36 ± 0.37	21.48 ± 1.18	19.99 ± 0.06
	3×8	21.87 ± 0.09	58.76 ± 21.65	44.37 ± 6.07	37.58 ± 8.31	27.67 ± 1.22
Qwen 3B	5×8	11.35 ± 0.01	11.35 ± 0.01	11.50 ± 0.00	11.39 ± 0.00	11.32 ± 0.01
	4×8	11.72 ± 0.02	22.32 ± 15.58	12.44 ± 0.07	12.24 ± 0.03	11.93 ± 0.01
	3×8	12.94 ± 0.01	283.97 ± 113.15	14.94 ± 0.08	14.32 ± 0.03	13.96 ± 0.00
Qwen 7B	5×8	10.61 ± 0.01	10.61 ± 0.01	10.80 ± 0.02	10.63 ± 0.02	10.51 ± 0.01
	4×8	10.94 ± 0.00	11.79 ± 0.16	11.66 ± 0.07	11.39 ± 0.01	10.93 ± 0.03
	3×8	11.85 ± 0.01	15.10 ± 0.19	13.91 ± 0.34	13.24 ± 0.07	12.27 ± 0.12
Qwen 14B	5×8	9.10 ± 0.00	9.10 ± 0.00	9.22 ± 0.01	9.26 ± 0.23	9.06 ± 0.00
	4×8	9.33 ± 0.01	9.77 ± 0.04	9.78 ± 0.01	9.71 ± 0.02	9.40 ± 0.00
	3×8	10.01 ± 0.02	12.02 ± 0.20	11.44 ± 0.47	10.82 ± 0.26	10.38 ± 0.01
Qwen 32B	5×8	8.84 ± 0.00	8.84 ± 0.00	8.89 ± 0.01	8.84 ± 0.01	8.79 ± 0.00
	4×8	8.98 ± 0.00	9.34 ± 0.05	9.25 ± 0.01	9.17 ± 0.02	8.98 ± 0.00
	3×8	9.46 ± 0.04	10.54 ± 0.20	10.10 ± 0.02	9.77 ± 0.01	9.49 ± 0.00

†AQLM (Ind.), AQLM (Drop), and DbyD (Uni.) are single runs; in training.

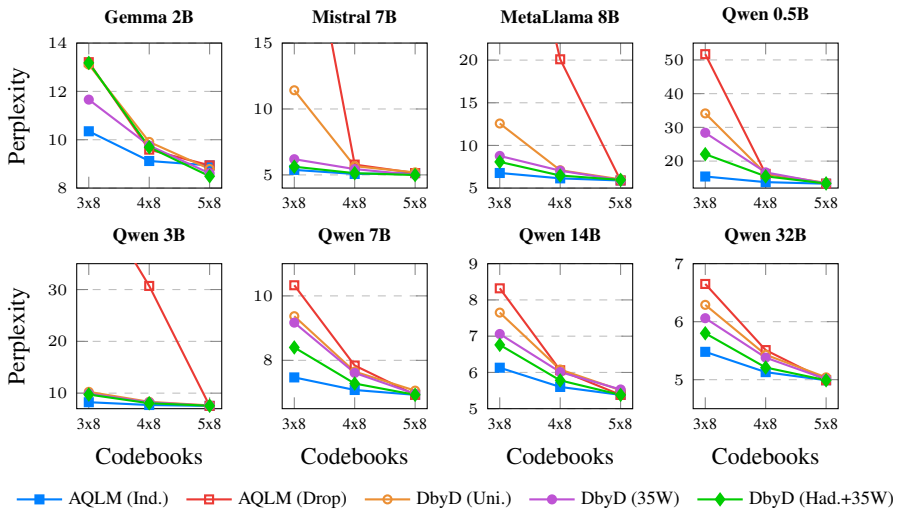


Figure 15: Perplexity on WIKITEXT2 (↓). Extreme outliers for AQLM (Drop) are clipped.

Table 8: Perplexity on WIKITEXT2 (↓). Values plotted in Figure 15.

Model	Code-books	AQLM (Ind.)	AQLM (Drop)	DbyD (Uni.)	DbyD (35W)	DbyD (Had.+35W)
Gemma 2B†	5 × 8	8.94	8.94	8.79	8.61 ± 0.12	8.49 ± 0.00
	4 × 8	9.12	9.59	9.91	9.76 ± 0.33	9.69 ± 0.31
	3 × 8	10.35	13.21	13.11	11.66 ± 0.48	13.18 ± 1.15
Mistral 7B	5 × 8	5.14 ± 0.26	5.14 ± 0.26	5.19 ± 0.24	5.00 ± 0.00	4.99 ± 0.00
	4 × 8	5.07 ± 0.01	5.78 ± 0.30	5.66 ± 0.24	5.44 ± 0.15	5.13 ± 0.00
	3 × 8	5.38 ± 0.08	112.75 ± 93.38	11.42 ± 5.14	6.19 ± 0.21	5.62 ± 0.02
MetaLlama 8B	5 × 8	5.89 ± 0.00	5.89 ± 0.00	6.02 ± 0.00	5.93 ± 0.00	5.96 ± 0.00
	4 × 8	6.14 ± 0.01	20.10 ± 20.10	7.09 ± 0.22	7.04 ± 0.12	6.47 ± 0.04
	3 × 8	6.77 ± 0.00	$4.6\times10^4 \pm 7.6\times10^4$	12.57 ± 1.10	8.77 ± 0.17	8.07 ± 0.05
Qwen 0.5B	5 × 8	13.30 ± 0.01	13.30 ± 0.01	13.54 ± 0.04	13.36 ± 0.02	13.36 ± 0.04
	4 × 8	13.76 ± 0.02	16.21 ± 0.34	16.29 ± 0.17	16.65 ± 0.70	15.49 ± 0.16
	3 × 8	15.43 ± 0.07	51.71 ± 24.90	34.14 ± 6.33	28.40 ± 5.83	22.05 ± 0.67
Qwen 3B	5 × 8	7.54 ± 0.00	7.54 ± 0.00	7.63 ± 0.01	7.57 ± 0.01	7.57 ± 0.00
	4 × 8	7.71 ± 0.01	30.70 ± 30.67	8.32 ± 0.07	8.22 ± 0.03	8.05 ± 0.03
	3 × 8	8.25 ± 0.01	365.00 ± 160.30	10.25 ± 0.09	9.91 ± 0.00	9.73 ± 0.11
Qwen 7B	5 × 8	6.93 ± 0.01	6.93 ± 0.01	7.06 ± 0.02	6.96 ± 0.01	6.93 ± 0.01
	4 × 8	7.08 ± 0.01	7.84 ± 0.02	7.65 ± 0.04	7.61 ± 0.05	7.28 ± 0.01
	3 × 8	7.47 ± 0.01	10.33 ± 0.22	9.37 ± 0.11	9.17 ± 0.02	8.40 ± 0.05
Qwen 14B	5 × 8	5.38 ± 0.00	5.38 ± 0.00	5.51 ± 0.01	5.53 ± 0.20	5.38 ± 0.00
	4 × 8	5.60 ± 0.01	6.06 ± 0.06	6.08 ± 0.03	6.01 ± 0.09	5.78 ± 0.02
	3 × 8	6.13 ± 0.00	8.32 ± 0.24	7.65 ± 0.38	7.06 ± 0.04	6.76 ± 0.04
Qwen 32B	5 × 8	4.99 ± 0.01	4.99 ± 0.01	5.04 ± 0.01	5.01 ± 0.00	4.99 ± 0.01
	4 × 8	5.13 ± 0.00	5.51 ± 0.01	5.43 ± 0.01	5.38 ± 0.02	5.21 ± 0.01
	3 × 8	5.48 ± 0.01	6.65 ± 0.20	6.29 ± 0.05	6.06 ± 0.01	5.80 ± 0.03

†AQLM (Ind.), AQLM (Drop), and DbyD (Uni.) are single runs; in training.

Table 9: Average zero-shot accuracy across 5 Tasks (↑). Values plotted in Figure 6.

Model	Code-books	AQLM (Ind.)	AQLM (Drop)	DbyD (Uni.)	DbyD (35W)	DbyD (Had.+35W)
Gemma 2B	5×8	65.59 ± 1.02	65.59 ± 1.02	64.62 ± 1.03	65.48 ± 1.03	65.13 ± 1.03
	4×8	63.95 ± 1.03	64.42 ± 1.03	62.38 ± 1.04	63.90 ± 1.03	63.48 ± 1.03
	3×8	60.37 ± 1.05	61.22 ± 1.05	56.51 ± 1.05	60.61 ± 1.05	60.17 ± 1.05
Mistral 7B	5×8	71.72 ± 1.05	71.72 ± 1.05	72.15 ± 1.04	72.85 ± 1.04	73.08 ± 0.97
	4×8	72.46 ± 1.04	69.94 ± 1.07	69.96 ± 1.07	70.81 ± 1.06	72.32 ± 1.00
	3×8	70.97 ± 1.05	44.84 ± 1.11	61.78 ± 1.10	67.48 ± 1.08	70.20 ± 1.02
MetaLlama 8B	5×8	72.42 ± 1.04	72.42 ± 1.04	71.62 ± 1.04	72.09 ± 1.05	73.37 ± 0.97
	4×8	71.00 ± 1.05	65.54 ± 1.08	68.52 ± 1.07	69.60 ± 1.06	72.36 ± 1.00
	3×8	70.00 ± 1.06	38.08 ± 1.09	55.80 ± 1.11	64.81 ± 1.09	67.81 ± 1.03
Qwen 0.5B	5×8	53.25 ± 1.12	53.25 ± 1.12	53.62 ± 1.12	53.86 ± 1.12	54.17 ± 1.12
	4×8	53.60 ± 1.11	52.80 ± 1.12	51.86 ± 1.12	53.14 ± 1.12	52.40 ± 1.12
	3×8	49.96 ± 1.12	44.02 ± 1.12	45.88 ± 1.12	48.47 ± 1.11	48.36 ± 1.12
Qwen 3B	5×8	67.83 ± 1.07	67.83 ± 1.07	68.18 ± 1.07	68.17 ± 1.07	68.11 ± 1.07
	4×8	66.68 ± 1.08	62.11 ± 1.10	65.85 ± 1.09	67.28 ± 1.08	67.80 ± 1.08
	3×8	65.48 ± 1.09	39.64 ± 1.10	62.53 ± 1.10	63.06 ± 1.10	64.32 ± 1.10
Qwen 7B	5×8	73.59 ± 1.04	73.59 ± 1.04	72.75 ± 1.05	73.72 ± 1.04	73.28 ± 1.04
	4×8	72.37 ± 1.05	71.40 ± 1.06	71.32 ± 1.06	72.52 ± 1.05	71.37 ± 1.06
	3×8	70.83 ± 1.07	66.77 ± 1.10	67.30 ± 1.09	69.54 ± 1.08	68.66 ± 1.08
Qwen 14B	5×8	76.85 ± 1.01	76.85 ± 1.01	76.03 ± 1.02	76.76 ± 1.01	77.20 ± 0.93
	4×8	75.94 ± 1.02	74.73 ± 1.03	75.80 ± 1.02	75.99 ± 1.01	76.63 ± 0.96
	3×8	74.89 ± 1.03	70.00 ± 1.07	73.92 ± 1.04	73.93 ± 1.04	75.07 ± 0.98
Qwen 32B	5×8	74.53 ± 1.04	74.53 ± 1.04	74.20 ± 1.04	74.74 ± 1.03	74.98 ± 1.03
	4×8	74.50 ± 1.04	73.10 ± 1.05	72.56 ± 1.05	73.98 ± 1.04	75.22 ± 1.03
	3×8	74.94 ± 1.03	71.08 ± 1.07	71.93 ± 1.06	72.94 ± 1.05	73.88 ± 1.04

H.2 Detailed Accuracy Performance Results: Qwen Family

In this section, we provide the detailed zero-shot accuracy breakdowns for the Qwen model family across five tasks: ARC Challenge, ARC Easy, HellaSwag, PIQA, and Winogrande.

Table 10: Zero-shot accuracy on arc_challenge (%) (↑)

Model	Code-books	AQLM (Ind.)	AQLM (Drop)	DbyD (Uni.)	DbyD (35W)	DbyD (Had.+35W)
Qwen 0.5B	5×8	32.34 ± 1.37	32.34 ± 1.37	32.85 ± 1.37	32.88 ± 1.37	33.11 ± 1.37
	4×8	32.39 ± 1.37	31.71 ± 1.36	32.20 ± 1.37	31.62 ± 1.36	31.31 ± 1.36
	3×8	28.81 ± 1.33	27.16 ± 1.30	27.11 ± 1.30	28.64 ± 1.32	29.12 ± 1.33
Qwen 3B	5×8	46.36 ± 1.46	46.36 ± 1.46	47.30 ± 1.46	46.70 ± 1.46	47.19 ± 1.46
	4×8	45.76 ± 1.46	41.90 ± 1.44	44.77 ± 1.45	47.13 ± 1.46	47.78 ± 1.46
	3×8	44.63 ± 1.45	23.26 ± 1.24	41.41 ± 1.44	40.90 ± 1.44	44.40 ± 1.45
Qwen 7B	5×8	56.46 ± 1.45	56.46 ± 1.45	55.69 ± 1.45	56.31 ± 1.45	54.55 ± 1.46
	4×8	55.12 ± 1.45	55.17 ± 1.45	54.10 ± 1.46	56.94 ± 1.45	53.13 ± 1.46
	3×8	54.44 ± 1.45	51.14 ± 1.46	49.57 ± 1.46	53.47 ± 1.46	49.89 ± 1.46
Qwen 14B	5×8	61.92 ± 1.42	61.92 ± 1.42	60.61 ± 1.43	62.17 ± 1.42	62.34 ± 1.41
	4×8	60.49 ± 1.43	58.02 ± 1.44	60.89 ± 1.43	60.84 ± 1.42	60.78 ± 1.43
	3×8	59.16 ± 1.44	51.20 ± 1.46	58.16 ± 1.44	58.82 ± 1.44	59.24 ± 1.44
Qwen 32B	5×8	58.05 ± 1.44	58.05 ± 1.44	57.08 ± 1.45	57.93 ± 1.44	58.59 ± 1.44
	4×8	57.17 ± 1.45	55.57 ± 1.45	54.18 ± 1.46	57.54 ± 1.44	59.36 ± 1.43
	3×8	58.16 ± 1.44	52.67 ± 1.46	54.44 ± 1.45	56.26 ± 1.45	57.82 ± 1.44

Table 11: Zero-shot accuracy on arc_easy (%) (↑)

Model	Code-books	AQLM (Ind.)	AQLM (Drop)	DbyD (Uni.)	DbyD (35W)	DbyD (Had.+35W)
Qwen 0.5B	5×8	57.00 ± 1.02	57.00 ± 1.02	60.35 ± 1.00	59.46 ± 1.01	59.82 ± 1.00
	4×8	60.06 ± 1.00	57.76 ± 1.01	57.43 ± 1.01	60.97 ± 1.00	56.76 ± 1.01
	3×8	53.65 ± 1.02	41.79 ± 1.02	48.81 ± 1.03	53.75 ± 1.02	49.35 ± 1.03
Qwen 3B	5×8	72.50 ± 0.92	72.50 ± 0.92	73.82 ± 0.90	73.91 ± 0.90	73.53 ± 0.90
	4×8	70.19 ± 0.94	64.56 ± 0.97	69.62 ± 0.94	72.78 ± 0.91	74.15 ± 0.90
	3×8	70.90 ± 0.93	38.06 ± 0.99	67.16 ± 0.96	67.33 ± 0.96	69.04 ± 0.95
Qwen 7B	5×8	81.23 ± 0.80	81.23 ± 0.80	79.04 ± 0.84	81.40 ± 0.80	80.56 ± 0.81
	4×8	78.77 ± 0.84	77.67 ± 0.85	76.17 ± 0.88	79.25 ± 0.83	75.94 ± 0.88
	3×8	75.76 ± 0.88	71.66 ± 0.92	69.76 ± 0.94	74.71 ± 0.89	72.96 ± 0.91
Qwen 14B	5×8	81.18 ± 0.80	81.18 ± 0.80	78.97 ± 0.84	80.50 ± 0.81	81.99 ± 0.79
	4×8	79.01 ± 0.84	77.33 ± 0.86	79.88 ± 0.82	80.12 ± 0.80	82.29 ± 0.78
	3×8	77.21 ± 0.86	69.95 ± 0.94	77.61 ± 0.85	77.40 ± 0.85	80.88 ± 0.81
Qwen 32B	5×8	75.59 ± 0.88	75.59 ± 0.88	75.17 ± 0.89	76.19 ± 0.87	77.16 ± 0.86
	4×8	75.79 ± 0.88	74.13 ± 0.90	72.01 ± 0.92	74.83 ± 0.89	78.17 ± 0.85
	3×8	77.03 ± 0.86	70.45 ± 0.94	73.36 ± 0.90	74.45 ± 0.89	77.19 ± 0.86

Table 12: Zero-shot accuracy on hellaswag (%) (↑)

Model	Code-books	AQLM (Ind.)	AQLM (Drop)	DbyD (Uni.)	DbyD (35W)	DbyD (Had.+35W)
Qwen 0.5B	5×8	51.57 ± 0.50	51.57 ± 0.50	50.42 ± 0.50	51.20 ± 0.50	51.48 ± 0.50
	4×8	49.78 ± 0.50	49.13 ± 0.50	47.34 ± 0.50	47.97 ± 0.50	49.21 ± 0.50
	3×8	45.65 ± 0.50	38.59 ± 0.49	39.21 ± 0.49	41.95 ± 0.49	43.50 ± 0.49
Qwen 3B	5×8	73.09 ± 0.44	73.09 ± 0.44	72.56 ± 0.45	72.85 ± 0.44	73.01 ± 0.44
	4×8	72.05 ± 0.45	64.62 ± 0.47	70.20 ± 0.46	71.15 ± 0.45	71.68 ± 0.45
	3×8	68.27 ± 0.46	29.61 ± 0.45	65.84 ± 0.47	67.19 ± 0.47	67.10 ± 0.47
Qwen 7B	5×8	80.00 ± 0.40	80.00 ± 0.40	79.62 ± 0.40	80.00 ± 0.40	80.17 ± 0.40
	4×8	79.46 ± 0.40	78.91 ± 0.41	78.72 ± 0.41	78.97 ± 0.41	79.45 ± 0.40
	3×8	77.30 ± 0.42	73.63 ± 0.44	75.35 ± 0.43	76.17 ± 0.43	76.74 ± 0.42
Qwen 14B	5×8	84.22 ± 0.36	84.22 ± 0.36	83.80 ± 0.37	83.93 ± 0.37	84.08 ± 0.36
	4×8	83.74 ± 0.37	83.66 ± 0.37	83.28 ± 0.37	83.38 ± 0.37	83.57 ± 0.37
	3×8	81.95 ± 0.38	80.13 ± 0.40	81.92 ± 0.38	81.31 ± 0.39	81.51 ± 0.39
Qwen 32B	5×8	85.11 ± 0.36	85.11 ± 0.36	84.82 ± 0.36	84.92 ± 0.36	84.91 ± 0.36
	4×8	84.90 ± 0.36	84.88 ± 0.36	84.36 ± 0.36	84.72 ± 0.36	84.79 ± 0.36
	3×8	83.79 ± 0.37	83.50 ± 0.37	82.40 ± 0.38	83.09 ± 0.37	83.48 ± 0.37

Table 13: Zero-shot accuracy on piqa (%) (↑)

Model	Code-books	AQLM (Ind.)	AQLM (Drop)	DbyD (Uni.)	DbyD (35W)	DbyD (Had.+35W)
Qwen 0.5B	5×8	69.48 ± 1.07	69.48 ± 1.07	69.10 ± 1.08	69.82 ± 1.07	70.09 ± 1.07
	4×8	69.47 ± 1.07	68.92 ± 1.08	67.18 ± 1.09	67.95 ± 1.09	68.42 ± 1.08
	3×8	66.14 ± 1.10	61.03 ± 1.14	62.02 ± 1.13	64.31 ± 1.11	65.07 ± 1.11
Qwen 3B	5×8	78.75 ± 0.95	78.75 ± 0.95	78.15 ± 0.96	78.71 ± 0.96	78.22 ± 0.96
	4×8	77.60 ± 0.97	74.61 ± 1.01	76.93 ± 0.98	77.64 ± 0.97	77.62 ± 0.97
	3×8	76.19 ± 0.99	55.84 ± 1.16	74.72 ± 1.01	75.10 ± 1.01	75.82 ± 1.00
Qwen 7B	5×8	80.10 ± 0.93	80.10 ± 0.93	79.67 ± 0.94	80.16 ± 0.93	80.01 ± 0.93
	4×8	79.22 ± 0.95	77.64 ± 0.97	78.18 ± 0.96	79.49 ± 0.94	79.44 ± 0.94
	3×8	77.79 ± 0.97	74.61 ± 1.01	76.30 ± 0.99	76.66 ± 0.99	76.95 ± 0.98
Qwen 14B	5×8	81.50 ± 0.90	81.50 ± 0.90	81.47 ± 0.91	81.61 ± 0.90	81.95 ± 0.90
	4×8	81.25 ± 0.91	80.87 ± 0.92	81.03 ± 0.91	80.96 ± 0.92	81.12 ± 0.91
	3×8	80.52 ± 0.93	80.05 ± 0.93	79.54 ± 0.94	80.16 ± 0.93	80.76 ± 0.92
Qwen 32B	5×8	81.26 ± 0.91	81.26 ± 0.91	81.10 ± 0.91	81.28 ± 0.91	81.10 ± 0.91
	4×8	80.90 ± 0.92	80.18 ± 0.93	79.60 ± 0.94	80.07 ± 0.93	80.96 ± 0.92
	3×8	81.17 ± 0.91	79.47 ± 0.94	78.75 ± 0.95	78.98 ± 0.95	79.69 ± 0.94

Table 14: Zero-shot accuracy on winogrande (%) (↑)

Model	Code-books	AQLM (Ind.)	AQLM (Drop)	DbyD (Uni.)	DbyD (35W)	DbyD (Had.+35W)
Qwen 0.5B	5×8	55.85 ± 1.39	55.85 ± 1.39	55.38 ± 1.40	55.96 ± 1.39	56.35 ± 1.40
	4×8	56.30 ± 1.39	56.48 ± 1.40	55.17 ± 1.40	57.17 ± 1.39	56.30 ± 1.39
	3×8	55.57 ± 1.40	51.54 ± 1.40	52.25 ± 1.40	53.70 ± 1.40	54.75 ± 1.40
Qwen 3B	5×8	68.46 ± 1.30	68.46 ± 1.30	69.06 ± 1.30	68.69 ± 1.30	68.61 ± 1.31
	4×8	67.80 ± 1.31	64.88 ± 1.34	67.72 ± 1.32	67.69 ± 1.32	67.77 ± 1.31
	3×8	67.40 ± 1.32	51.43 ± 1.40	63.51 ± 1.35	64.77 ± 1.34	65.25 ± 1.34
Qwen 7B	5×8	70.14 ± 1.28	70.14 ± 1.28	69.75 ± 1.29	70.72 ± 1.28	71.12 ± 1.28
	4×8	69.27 ± 1.30	67.61 ± 1.31	69.43 ± 1.29	67.93 ± 1.31	68.88 ± 1.30
	3×8	68.85 ± 1.30	62.83 ± 1.36	65.54 ± 1.34	66.69 ± 1.32	66.74 ± 1.33
Qwen 14B	5×8	75.45 ± 1.21	75.45 ± 1.21	75.30 ± 1.21	75.59 ± 1.21	75.64 ± 1.20
	4×8	75.19 ± 1.21	73.77 ± 1.23	73.93 ± 1.23	74.64 ± 1.22	75.40 ± 1.21
	3×8	75.61 ± 1.21	68.69 ± 1.30	72.37 ± 1.25	71.98 ± 1.26	72.95 ± 1.25
Qwen 32B	5×8	72.64 ± 1.25	72.64 ± 1.25	72.85 ± 1.25	73.40 ± 1.24	73.14 ± 1.25
	4×8	73.74 ± 1.24	70.72 ± 1.28	72.66 ± 1.25	72.72 ± 1.25	72.85 ± 1.25
	3×8	74.53 ± 1.22	69.32 ± 1.29	70.69 ± 1.28	71.90 ± 1.26	71.24 ± 1.27

H.3 Detailed Accuracy Performance Results: Gemma, Mistral, and Llama

In this section, we provide the detailed zero-shot accuracy breakdowns for the select Gemma, Mistral, and Llama models across five tasks: ARC Challenge, ARC Easy, HellaSwag, PIQA, and Winogrande.

Table 15: Zero-shot accuracy on arc_challenge (%) (↑)

Model	Code-books	AQLM (Ind.)	AQLM (Drop)	DbyD (Uni.)	DbyD (35W)	DbyD (Had.+35W)
Gemma 2B	5×8	41.89 ± 1.44	41.89 ± 1.44	41.64 ± 1.44	42.24 ± 1.44	40.87 ± 1.44
	4×8	40.27 ± 1.43	41.64 ± 1.44	38.31 ± 1.42	40.27 ± 1.43	39.22 ± 1.43
	3×8	35.49 ± 1.40	38.91 ± 1.42	32.17 ± 1.37	36.83 ± 1.41	36.32 ± 1.40
Mistral 7B	5×8	50.28 ± 1.46	50.28 ± 1.46	50.45 ± 1.46	52.02 ± 1.46	51.59 ± 1.46
	4×8	50.97 ± 1.46	48.30 ± 1.46	48.69 ± 1.46	49.12 ± 1.46	50.54 ± 1.46
	3×8	49.09 ± 1.46	26.85 ± 1.29	40.93 ± 1.43	44.31 ± 1.45	47.18 ± 1.46
MetaLlama 8B	5×8	52.19 ± 1.46	52.19 ± 1.46	50.08 ± 1.46	51.82 ± 1.46	53.47 ± 1.46
	4×8	49.72 ± 1.46	43.83 ± 1.45	46.73 ± 1.46	48.09 ± 1.46	52.30 ± 1.46
	3×8	49.20 ± 1.46	23.15 ± 1.23	33.47 ± 1.38	41.67 ± 1.44	45.48 ± 1.45

Table 16: Zero-shot accuracy on arc_easy (%) (↑)

Model	Code-books	AQLM (Ind.)	AQLM (Drop)	DbyD (Uni.)	DbyD (35W)	DbyD (Had.+35W)
Gemma 2B	5×8	72.56 ± 0.92	72.56 ± 0.92	71.34 ± 0.93	72.11 ± 0.92	71.62 ± 0.93
	4×8	68.86 ± 0.95	70.08 ± 0.94	67.34 ± 0.96	69.61 ± 0.94	69.78 ± 0.94
	3×8	63.85 ± 0.99	65.28 ± 0.98	53.41 ± 1.02	64.31 ± 0.98	65.24 ± 0.98
Mistral 7B	5×8	76.49 ± 0.87	76.49 ± 0.87	76.57 ± 0.87	77.51 ± 0.86	78.27 ± 0.84
	4×8	78.70 ± 0.84	74.82 ± 0.89	75.14 ± 0.89	75.17 ± 0.89	77.32 ± 0.86
	3×8	76.99 ± 0.86	43.04 ± 1.00	65.02 ± 0.98	72.47 ± 0.92	75.19 ± 0.89
MetaLlama 8B	5×8	76.50 ± 0.87	76.50 ± 0.87	75.62 ± 0.88	75.98 ± 0.88	80.89 ± 0.81
	4×8	74.19 ± 0.90	68.80 ± 0.95	70.87 ± 0.93	72.22 ± 0.92	78.89 ± 0.84
	3×8	73.76 ± 0.90	29.57 ± 0.94	51.60 ± 1.01	65.77 ± 0.97	72.59 ± 0.92

Table 17: Zero-shot accuracy on hellaswag (%) (↑)

Model	Code-books	AQLM (Ind.)	AQLM (Drop)	DbyD (Uni.)	DbyD (35W)	DbyD (Had.+35W)
Gemma 2B	5×8	70.87 ± 0.45	70.87 ± 0.45	68.59 ± 0.46	70.59 ± 0.45	70.73 ± 0.45
	4×8	69.18 ± 0.46	69.27 ± 0.46	66.96 ± 0.47	68.99 ± 0.46	68.70 ± 0.46
	3×8	64.52 ± 0.48	63.65 ± 0.48	62.26 ± 0.48	64.94 ± 0.48	63.48 ± 0.48
Mistral 7B	5×8	79.87 ± 0.40	79.87 ± 0.40	79.89 ± 0.40	80.28 ± 0.40	80.29 ± 0.40
	4×8	79.95 ± 0.40	77.65 ± 0.42	78.60 ± 0.41	79.14 ± 0.41	79.73 ± 0.40
	3×8	77.92 ± 0.41	39.56 ± 0.48	66.14 ± 0.47	75.81 ± 0.43	77.67 ± 0.42
MetaLlama 8B	5×8	78.86 ± 0.41	78.86 ± 0.41	78.19 ± 0.41	78.89 ± 0.41	78.58 ± 0.41
	4×8	77.83 ± 0.41	69.66 ± 0.45	76.15 ± 0.43	76.35 ± 0.42	77.97 ± 0.41
	3×8	75.91 ± 0.43	31.47 ± 0.46	63.93 ± 0.48	72.82 ± 0.44	73.35 ± 0.44

Table 18: Zero-shot accuracy on piqa (%) (↑)

Model	Code-books	AQLM (Ind.)	AQLM (Drop)	DbyD (Uni.)	DbyD (35W)	DbyD (Had.+35W)
Gemma 2B	5×8	77.58 ± 0.97	77.58 ± 0.97	77.42 ± 0.98	77.62 ± 0.97	77.91 ± 0.97
	4×8	77.04 ± 0.98	76.77 ± 0.99	76.17 ± 0.99	76.53 ± 0.99	77.06 ± 0.98
	3×8	74.92 ± 1.01	75.35 ± 1.01	74.43 ± 1.02	74.94 ± 1.01	74.90 ± 1.01
Mistral 7B	5×8	80.76 ± 0.92	80.76 ± 0.92	81.59 ± 0.90	81.35 ± 0.91	81.66 ± 0.90
	4×8	81.07 ± 0.91	79.61 ± 0.94	79.87 ± 0.93	80.29 ± 0.93	81.40 ± 0.90
	3×8	80.38 ± 0.92	62.80 ± 1.12	75.28 ± 1.01	77.84 ± 0.97	80.11 ± 0.93
MetaLlama 8B	5×8	80.36 ± 0.93	80.36 ± 0.93	80.05 ± 0.93	80.05 ± 0.93	80.70 ± 0.92
	4×8	79.40 ± 0.95	75.24 ± 1.01	78.45 ± 0.96	78.91 ± 0.95	80.31 ± 0.93
	3×8	78.46 ± 0.96	53.77 ± 1.16	67.39 ± 1.09	74.66 ± 1.02	77.97 ± 0.97

Table 19: Zero-shot accuracy on winogrande (%) (↑)

Model	Code-books	AQLM (Ind.)	AQLM (Drop)	DbyD (Uni.)	DbyD (35W)	DbyD (Had.+35W)
Gemma 2B	5×8	65.04 ± 1.34	65.04 ± 1.34	64.09 ± 1.35	64.85 ± 1.34	64.54 ± 1.34
	4×8	64.40 ± 1.35	64.33 ± 1.35	63.14 ± 1.36	64.09 ± 1.35	62.64 ± 1.36
	3×8	63.06 ± 1.36	62.90 ± 1.36	60.30 ± 1.38	62.04 ± 1.36	60.93 ± 1.37
Mistral 7B	5×8	71.19 ± 1.27	71.19 ± 1.27	72.27 ± 1.26	73.11 ± 1.25	73.59 ± 1.24
	4×8	71.61 ± 1.27	69.30 ± 1.30	67.51 ± 1.32	70.35 ± 1.29	72.61 ± 1.25
	3×8	70.45 ± 1.28	51.96 ± 1.40	61.54 ± 1.36	66.95 ± 1.32	70.82 ± 1.27
MetaLlama 8B	5×8	74.19 ± 1.23	74.19 ± 1.23	74.17 ± 1.23	73.72 ± 1.24	73.22 ± 1.24
	4×8	73.85 ± 1.24	70.16 ± 1.28	70.40 ± 1.28	72.43 ± 1.25	72.35 ± 1.26
	3×8	72.67 ± 1.25	52.46 ± 1.40	62.61 ± 1.36	69.14 ± 1.30	69.67 ± 1.29

I QUANTITATIVE SOTA RESULTS

I.1 PERPLEXITY VALUES FOR THE SOTA METHODS

This section compares the proposed Drop-by-Drop (DbyD) method, specifically the DbyD (Had.+35W) variant, against the closest state-of-the-art post-training quantization methods that support multi-bitwidth inference, namely AnyBCQ, Any Precision LLM (AnyPrecLLM), and MatGPTQ Park et al. (2026; 2024); Kleinegger et al. (2026). All methods are evaluated at average precisions of 3, 4, and 5 bits on the WIKITEXT2 and C4 benchmarks across a diverse model architectures and scales. Tables 21 and 20 report the mean perplexity and standard deviation over *three* independent runs.

Across the QWEN family, DbyD achieves the best perplexity at every evaluated precision. On the broader collection of model families, DbyD consistently outperforms all three baselines on MISTRAL-7B-V0.3 and META-LLAMA-3-8B across all evaluated precisions. On GEMMA-2B, DbyD achieves the lowest C4 perplexity at every precision, while AnyPrecLLM achieves lower WIKITEXT2 perplexity at 3 bits.

Let us clarify certain notation in these tables. The *nan* entries for QWEN2.5-7B arise because AnyBCQ's quantization, run with the authors' default settings, produced a NaN reconstruction loss in the final decoder layer; the resulting checkpoint yields NaN perplexity and chance-level zero-shot accuracy. MatGPTQ entries for QWEN2.5-7B are omitted (denoted as –): Cholesky factorization of the layer-wise Hessian failed to converge even under increased damping (`rel_damp`), and we were unable to obtain a quantized model at this scale. Also, for QWEN2.5-32B, AnyBCQ was quantized with 1,500 reconstruction iterations rather than the default 5,000, since we observed that the reconstruction loss consistently plateaued before 1,500 iterations. Runs at the default setting terminated during intermediate-layer reconstruction, whereas the reduced configuration completed successfully.

Table 20: State-of-the-art perplexity comparison (↓) on C4. Values condensed in Table 1.

Method	Bits	Qwen 0.5B	Qwen 3B	Qwen 7B	Qwen 14B	Qwen 32B
DbyD (Had.+35W)	5	**17.87 ± 0.00**	**11.32 ± 0.01**	**10.51 ± 0.01**	**9.06 ± 0.00**	**8.79 ± 0.00**
AnyBCQ		24.70 ± 0.00	14.96 ± 0.00	*nan*	11.96 ± 0.00	11.63 ± 0.00
AnyPrecLLM		22.44 ± 0.02	13.77 ± 0.01	12.93 ± 0.03	11.01 ± 0.00	10.70 ± 0.00
MatGPTQ		23.25 ± 0.00	13.62 ± 0.00	–	11.07 ± 0.00	10.59 ± 0.00
DbyD (Had.+35W)	4	**19.99 ± 0.06**	**11.93 ± 0.01**	**10.93 ± 0.03**	**9.40 ± 0.00**	**8.98 ± 0.00**
AnyBCQ		25.89 ± 0.00	15.36 ± 0.00	*nan*	12.15 ± 0.00	11.78 ± 0.00
AnyPrecLLM		24.15 ± 0.10	14.26 ± 0.01	13.33 ± 0.08	11.28 ± 0.01	10.89 ± 0.01
MatGPTQ		26.72 ± 0.00	14.22 ± 0.00	–	11.47 ± 0.00	10.81 ± 0.00
DbyD (Had.+35W)	3	**27.67 ± 1.22**	**13.96 ± 0.00**	**12.27 ± 0.12**	**10.38 ± 0.01**	**9.49 ± 0.00**
AnyBCQ		30.62 ± 0.00	16.86 ± 0.00	*nan*	12.98 ± 0.00	12.30 ± 0.00
AnyPrecLLM		34.78 ± 0.33	16.71 ± 0.08	15.52 ± 0.68	12.44 ± 0.05	11.64 ± 0.03
MatGPTQ		80.06 ± 0.00	134.12 ± 0.00	–	13.23 ± 0.00	11.90 ± 0.00

Method	Bits	Gemma 2B	Mistral 7B	MetaLlama 8B
DbyD (Had.+35W)	5	**11.79 ± 0.00**	**7.54 ± 0.00**	**8.63 ± 0.00**
AnyBCQ		25.66 ± 0.00	17.48 ± 0.00	10.24 ± 0.00
AnyPrecLLM		13.67 ± 0.01	8.52 ± 0.00	9.68 ± 0.00
MatGPTQ		22.62 ± 0.00	9.18 ± 0.00	9.71 ± 0.00
DbyD (Had.+35W)	4	**13.23 ± 0.37**	**7.71 ± 0.00**	**9.38 ± 0.04**
AnyBCQ		25.85 ± 0.00	14.28 ± 0.00	10.81 ± 0.00
AnyPrecLLM		14.40 ± 0.05	8.67 ± 0.01	10.28 ± 0.00
MatGPTQ		23.25 ± 0.00	9.95 ± 0.00	10.60 ± 0.00
DbyD (Had.+35W)	3	**17.07 ± 1.15**	**8.32 ± 0.00**	**11.53 ± 0.05**
AnyBCQ		27.79 ± 0.00	10.49 ± 0.00	13.34 ± 0.00
AnyPrecLLM		17.58 ± 0.04	9.39 ± 0.03	13.14 ± 0.03
MatGPTQ		52.50 ± 0.00	11.95 ± 0.00	14.93 ± 0.00

Table 21: State-of-the-art perplexity comparison (↓) on WIKITEXT2.

Method	Bits	Qwen 0.5B	Qwen 3B	Qwen 7B	Qwen 14B	Qwen 32B
DbyD (Had.+35W)	5	**13.36 ± 0.04**	**7.57 ± 0.00**	**6.93 ± 0.01**	**5.38 ± 0.00**	**4.99 ± 0.01**
AnyBCQ		17.46 ± 0.00	9.56 ± 0.00	*nan*	7.25 ± 0.00	6.90 ± 0.00
AnyPrecLLM		14.66 ± 0.04	8.14 ± 0.01	7.54 ± 0.02	5.79 ± 0.01	5.38 ± 0.01
MatGPTQ		15.77 ± 0.00	8.34 ± 0.00	–	5.91 ± 0.00	5.42 ± 0.00
DbyD (Had.+35W)	4	**15.49 ± 0.16**	**8.05 ± 0.03**	**7.28 ± 0.01**	**5.78 ± 0.02**	**5.21 ± 0.01**
AnyBCQ		18.41 ± 0.00	9.84 ± 0.00	*nan*	7.45 ± 0.00	7.04 ± 0.00
AnyPrecLLM		16.08 ± 0.15	8.51 ± 0.03	7.75 ± 0.08	6.09 ± 0.01	5.60 ± 0.01
MatGPTQ		18.11 ± 0.00	8.88 ± 0.00	–	6.27 ± 0.00	5.64 ± 0.00
DbyD (Had.+35W)	3	**22.05 ± 0.67**	**9.73 ± 0.11**	**8.40 ± 0.05**	**6.76 ± 0.04**	**5.80 ± 0.03**
AnyBCQ		23.95 ± 0.00	11.19 ± 0.00	*nan*	8.18 ± 0.00	7.53 ± 0.00
AnyPrecLLM		24.87 ± 0.68	10.38 ± 0.04	9.02 ± 0.01	7.16 ± 0.05	6.34 ± 0.06
MatGPTQ		69.56 ± 0.00	129.00 ± 0.00	–	7.85 ± 0.00	6.70 ± 0.00

Method	Bits	Gemma 2B	Mistral 7B	MetaLlama 8B
DbyD (Had.+35W)	5	**8.49 ± 0.00**	**4.99 ± 0.00**	**5.96 ± 0.00**
AnyBCQ		40.80 ± 0.00	25.84 ± 0.00	7.22 ± 0.00
AnyPrecLLM		9.10 ± 0.01	5.35 ± 0.00	6.25 ± 0.00
MatGPTQ		15.77 ± 0.00	5.94 ± 0.00	6.41 ± 0.00
DbyD (Had.+35W)	4	**9.69 ± 0.31**	**5.13 ± 0.00**	**6.47 ± 0.04**
AnyBCQ		43.59 ± 0.00	16.95 ± 0.00	7.52 ± 0.00
AnyPrecLLM		9.72 ± 0.01	5.45 ± 0.01	6.60 ± 0.01
MatGPTQ		16.27 ± 0.00	6.86 ± 0.00	6.83 ± 0.00
DbyD (Had.+35W)	3	13.18 ± 1.15	**5.62 ± 0.02**	**8.07 ± 0.05**
AnyBCQ		48.95 ± 0.00	10.47 ± 0.00	8.87 ± 0.00
AnyPrecLLM		**12.52 ± 0.10**	5.98 ± 0.03	8.32 ± 0.02
MatGPTQ		42.44 ± 0.00	8.44 ± 0.00	9.70 ± 0.00

The similar pattern of DbyD outperforming also holds when each baseline is instead quantized over its own native bitwidth range, namely 2-4 bits for AnyBCQ and 3-8 bits for AnyPrecLLM, which we quantize, evaluate and report in Table 22 and Table 23. The entries marked with an asterisk (*) for the 14B and 32B models use the officially released QWEN3-14B and QWEN3-32B checkpoints, as we wanted to utilize the only QWEN variants released by the authors.

Moreover, quantizing QWEN2.5-32B for AnyPrecLLM and MatGPTQ, and both QWEN2.5-14B and QWEN2.5-32B for AnyBCQ, required H100 GPUs rather than the L40S GPUs used for the remaining models, since these methods required a larger memory capacity for the larger models. Whereas, Drop-by-Drop did not and was quantized on L40S GPUs throughout.

Table 22: SOTA perplexity comparison (↓) on C4 in native bitwidth ranges.

Method	Bits	Qwen 0.5B	Qwen 3B	Qwen 7B	Qwen 14B	Qwen 32B
DbyD (Had.+35W)	5	**17.87 ± 0.00**	**11.32 ± 0.01**	**10.51 ± 0.01**	**9.06 ± 0.00**	**8.79 ± 0.00**
AnyBCQ		–	–	–	–	–
AnyPrecLLM		22.45 ± 0.01	13.78 ± 0.01	12.93 ± 0.02	11.01 ± 0.01	10.70 ± 0.00
MatGPTQ		23.25 ± 0.00	13.62 ± 0.00	–	11.07 ± 0.00	10.59 ± 0.00
DbyD (Had.+35W)	4	**19.99 ± 0.06**	**11.93 ± 0.01**	**10.93 ± 0.03**	**9.40 ± 0.00**	**8.98 ± 0.00**
AnyBCQ		26.44 ± 0.00	15.54 ± 0.00	*nan*	14.66* ± 0.00	13.06* ± 0.00
AnyPrecLLM		24.23 ± 0.02	14.26 ± 0.02	13.34 ± 0.04	11.30 ± 0.00	10.88 ± 0.01
MatGPTQ		26.72 ± 0.00	14.22 ± 0.00	–	11.47 ± 0.00	10.81 ± 0.00
DbyD (Had.+35W)	3	**27.67 ± 1.22**	**13.96 ± 0.00**	**12.27 ± 0.12**	**10.38 ± 0.01**	**9.49 ± 0.00**
AnyBCQ		32.27 ± 0.00	17.17 ± 0.00	*nan*	15.51* ± 0.00	13.61* ± 0.00
AnyPrecLLM		35.33 ± 0.26	16.70 ± 0.06	15.44 ± 0.21	12.47 ± 0.01	11.63 ± 0.02
MatGPTQ		80.06 ± 0.00	134.12 ± 0.00	–	13.23 ± 0.00	11.90 ± 0.00

Method	Bits	Gemma 2B	Mistral 7B	MetaLlama 8B
DbyD (Had.+35W)	5	**11.79 ± 0.00**	**7.54 ± 0.00**	**8.63 ± 0.00**
AnyBCQ		–	–	–
AnyPrecLLM		13.65 ± 0.01	8.82 ± 0.50	9.74 ± 0.04
MatGPTQ		22.62 ± 0.00	9.18 ± 0.00	9.71 ± 0.00
DbyD (Had.+35W)	4	**13.23 ± 0.37**	**7.71 ± 0.00**	**9.38 ± 0.04**
AnyBCQ		26.78 ± 0.00	11.46 ± 0.00	10.55 ± 0.00
AnyPrecLLM		14.36 ± 0.01	8.67 ± 0.00	10.34 ± 0.05
MatGPTQ		23.25 ± 0.00	9.95 ± 0.00	10.60 ± 0.00
DbyD (Had.+35W)	3	**17.07 ± 1.15**	**8.32 ± 0.00**	**11.53 ± 0.05**
AnyBCQ		29.41 ± 0.00	12.11 ± 0.00	12.41 ± 0.00
AnyPrecLLM		17.55 ± 0.24	9.09 ± 0.49	13.05 ± 0.10
MatGPTQ		52.50 ± 0.00	11.95 ± 0.00	14.93 ± 0.00

Table 23: SOTA perplexity comparison (↓) on WIKITEXT2 in native bitwidth ranges.

Method	Bits	Qwen 0.5B	Qwen 3B	Qwen 7B	Qwen 14B	Qwen 32B
DbyD (Had.+35W)	5	**13.36 ± 0.04**	**7.57 ± 0.00**	**6.93 ± 0.01**	**5.38 ± 0.00**	**4.99 ± 0.01**
AnyBCQ		–	–	–	–	–
AnyPrecLLM		14.68 ± 0.03	8.14 ± 0.01	7.55 ± 0.02	5.79 ± 0.00	5.39 ± 0.01
MatGPTQ		15.77 ± 0.00	8.34 ± 0.00	–	5.91 ± 0.00	5.42 ± 0.00
DbyD (Had.+35W)	4	**15.49 ± 0.16**	**8.05 ± 0.03**	**7.28 ± 0.01**	**5.78 ± 0.02**	**5.21 ± 0.01**
AnyBCQ		19.11 ± 0.00	10.04 ± 0.00	*nan*	9.02* ± 0.00	7.81* ± 0.00
AnyPrecLLM		16.16 ± 0.14	8.49 ± 0.02	7.77 ± 0.04	6.09 ± 0.03	5.61 ± 0.02
MatGPTQ		18.11 ± 0.00	8.88 ± 0.00	–	6.27 ± 0.00	5.64 ± 0.00
DbyD (Had.+35W)	3	**22.05 ± 0.67**	**9.73 ± 0.11**	**8.40 ± 0.05**	**6.76 ± 0.04**	**5.80 ± 0.03**
AnyBCQ		26.20 ± 0.00	11.45 ± 0.00	*nan*	9.95* ± 0.00	8.42* ± 0.00
AnyPrecLLM		25.52 ± 0.68	10.33 ± 0.10	9.30 ± 0.23	7.12 ± 0.01	6.35 ± 0.01
MatGPTQ		69.56 ± 0.00	129.00 ± 0.00	–	7.85 ± 0.00	6.70 ± 0.00

Method	Bits	Gemma 2B	Mistral 7B	MetaLlama 8B
DbyD (Had.+35W)	5	**8.49 ± 0.00**	**4.99 ± 0.00**	**5.96 ± 0.00**
AnyBCQ		–	–	–
AnyPrecLLM		9.09 ± 0.01	5.35 ± 0.00	6.31 ± 0.05
MatGPTQ		15.77 ± 0.00	5.94 ± 0.00	6.41 ± 0.00
DbyD (Had.+35W)	4	9.69 ± 0.31	**5.13 ± 0.00**	**6.47 ± 0.04**
AnyBCQ		45.17 ± 0.00	12.93 ± 0.00	6.75 ± 0.00
AnyPrecLLM		**9.68 ± 0.03**	5.45 ± 0.00	6.64 ± 0.04
MatGPTQ		16.27 ± 0.00	6.86 ± 0.00	6.83 ± 0.00
DbyD (Had.+35W)	3	13.18 ± 1.15	**5.62 ± 0.02**	**8.07 ± 0.05**
AnyBCQ		54.33 ± 0.00	13.86 ± 0.00	8.20 ± 0.00
AnyPrecLLM		**12.29 ± 0.30**	5.99 ± 0.01	8.25 ± 0.07
MatGPTQ		42.44 ± 0.00	8.44 ± 0.00	9.70 ± 0.00

I.2 Zero-shot Accuracy Values for SOTA Methods

Table 24 reports average zero-shot accuracy over five tasks. DbyD achieves accuracy comparable to the state-of-the-art baselines, staying within 1.5 points of the best method in most settings, which is within one standard error of the difference between the two. At the same time, DbyD provides benefits over these baselines in perplexity, inference latency, and storage footprint, as detailed in Appendices I.1 and I.3, and Figure 7.

Table 24: Average zero-shot accuracy across 5 Tasks (↑), compared against SOTA methods.

Model	Bits	AQLM (Ind.)	DbyD (35W)	DbyD (Had.+35W)	AnyBCQ	AnyPrec	MatGPTQ
Gemma 2B	5	65.59 ± 1.02	65.48 ± 1.03	65.13 ± 1.03	64.50 ± 1.03	65.22 ± 1.03	64.37 ± 1.03
	4	63.95 ± 1.03	63.90 ± 1.03	63.48 ± 1.03	63.57 ± 1.03	64.88 ± 1.02	63.47 ± 1.04
	3	60.37 ± 1.05	60.61 ± 1.05	60.17 ± 1.05	60.94 ± 1.04	60.30 ± 1.04	56.22 ± 1.07
Mistral 7B	5	71.72 ± 1.05	72.85 ± 1.04	73.08 ± 0.97	58.11 ± 1.07	74.24 ± 0.96	73.00 ± 0.97
	4	72.46 ± 1.04	70.81 ± 1.06	72.32 ± 1.00	63.65 ± 1.05	73.59 ± 0.96	72.88 ± 0.97
	3	70.97 ± 1.05	67.48 ± 1.08	70.20 ± 1.02	70.43 ± 0.99	72.14 ± 0.98	70.25 ± 1.00
MetaLlama 8B	5	72.42 ± 1.04	72.09 ± 1.05	73.37 ± 0.97	72.90 ± 0.97	72.35 ± 0.98	72.12 ± 0.98
	4	71.00 ± 1.05	69.60 ± 1.06	72.36 ± 1.00	72.33 ± 0.98	71.41 ± 0.98	71.72 ± 0.98
	3	70.00 ± 1.06	64.81 ± 1.09	67.81 ± 1.03	69.25 ± 1.00	68.25 ± 1.01	64.15 ± 1.03
Qwen 0.5B	5	53.25 ± 1.12	53.86 ± 1.12	54.17 ± 1.12	54.76 ± 1.07	54.60 ± 1.07	52.89 ± 1.07
	4	53.60 ± 1.11	53.14 ± 1.12	52.40 ± 1.12	53.79 ± 1.07	54.67 ± 1.07	51.98 ± 1.07
	3	49.96 ± 1.12	48.47 ± 1.11	48.36 ± 1.12	50.82 ± 1.07	48.28 ± 1.06	44.69 ± 1.07
Qwen 3B	5	67.83 ± 1.07	68.17 ± 1.07	68.11 ± 1.07	68.21 ± 1.01	68.18 ± 1.01	68.02 ± 1.01
	4	66.68 ± 1.08	67.28 ± 1.08	67.80 ± 1.08	68.12 ± 1.01	68.58 ± 1.01	67.86 ± 1.01
	3	65.48 ± 1.09	63.06 ± 1.10	64.32 ± 1.10	65.19 ± 1.03	65.32 ± 1.03	44.47 ± 1.07
Qwen 7B	5	73.59 ± 1.04	73.72 ± 1.04	73.28 ± 1.04	$34.38^{\dagger} \pm 1.02$	73.43 ± 0.97	–
	4	72.37 ± 1.05	72.52 ± 1.05	71.37 ± 1.06	$34.38^{\dagger} \pm 1.02$	72.14 ± 0.99	–
	3	70.83 ± 1.07	69.54 ± 1.08	68.66 ± 1.08	$34.38^{\dagger} \pm 1.02$	70.13 ± 1.00	–
Qwen 14B	5	76.85 ± 1.01	76.76 ± 1.01	77.20 ± 0.93	76.76 ± 0.94	76.92 ± 0.94	76.72 ± 0.94
	4	75.94 ± 1.02	75.99 ± 1.01	76.63 ± 0.96	76.04 ± 0.95	76.04 ± 0.95	76.19 ± 0.95
	3	74.89 ± 1.03	73.93 ± 1.04	75.07 ± 0.98	75.78 ± 0.95	73.16 ± 0.98	72.33 ± 0.99
Qwen 32B	5	74.53 ± 1.04	74.74 ± 1.03	74.98 ± 1.03	$74.68^{\ddagger} \pm 0.97$	74.83 ± 0.97	75.02 ± 0.96
	4	74.50 ± 1.04	73.98 ± 1.04	75.22 ± 1.03	$74.57^{\ddagger} \pm 0.97$	74.44 ± 0.97	74.49 ± 0.97
	3	74.94 ± 1.03	72.94 ± 1.05	73.88 ± 1.04	$74.92^{\ddagger} \pm 0.96$	74.13 ± 0.97	72.87 ± 0.99

† AnyBCQ quantization of Qwen2.5-7B diverged (NaN loss); accuracy is at chance level.
‡ AnyBCQ Qwen2.5-32B trained for 1,500 instead of 5,000 iterations.
– MatGPTQ quantization of Qwen2.5-7B did not converge; no results reported.

I.3 Latency Values for the SOTA Methods

Motivation. AQLM has specialized CUDA kernels only for the 1×16 and 2×8 codebook configurations. Their 2×8 kernel keeps 8 copies of each codebook in shared memory, which does not fit within the per-block limit once more codebooks are added. As a result, for 3–5 codebooks, AQLM falls back to a much slower generic path: a Triton GEMV for 4 codebooks, and a full dequantize-then-GEMM for 3 and 5, since Triton requires power-of-two ranges. We, therefore, wrote `codeNx8`, a kernel for N 8-bit codebooks over groups of 8 weights, i.e. N bits per weight. It uses AQLM's quantized weights (codes, codebooks and per-row scales) as they are; the quantized model is unchanged.

Kernel design. A codebook-quantized weight group is the sum of several 8-value codebook vectors. The usual approach rebuilds those weights before taking the dot product with the input. Because the dot product is linear, at batch size 1 `codeNx8` instead computes the dot product of the input with every codebook entry once, then replaces each weight with a scalar table lookup. The main advances are:

- **Table lookups replace weight reconstruction.** Each code costs one small shared-memory read instead of fetching a codebook vector and performing 8 multiply-adds. The table depends on the input, so it is rebuilt on every call, and all reported latencies include it.
- **Contiguous memory reads.** Codes are reordered once at load time so that each group of 32 threads reads one contiguous block of memory.
- **Work split across the GPU.** Long rows are divided among many thread blocks, so layers with few output rows still spread across many of the GPU's multiprocessors. Partial results are combined within the same kernel launch in a fixed order, so outputs are deterministic.

Experimental setup. All kernels were benchmarked on a single NVIDIA L40S GPU (48 GB memory, 96 MB L2 cache). `codeNx8` was built with CUDA 12.6 and PyTorch 2.7.1. We evaluate batch-1 fp16 matrix-vector products, the core operation of autoregressive decoding.

We compare against dense FP16 cuBLAS and three SOTA quantized GEMV kernels: AnyBCQ, AnyPrecisionLLM and MatGPTQ. Each baseline is called through its authors' unmodified kernel code, in the software environment the authors provide, at the same bit width as ours. MatGPTQ has no 5-bit kernel, so it covers only the 3- and 4-bit settings. All methods were run in the same job on the same GPU, with identical shapes and timing procedure.

We measured latency in two settings: with weights allowed to stay in the L2 cache between calls, and with the cache cleared before every call. `codeNx8` was faster than every baseline on average in both settings. We report the second setting because it matches decoding: the quantized weights of every model we evaluate exceed the L2 cache, so each layer's weights are read from DRAM once per generated token. Each latency is the mean over 3 repetitions of 100 timed calls.

Results. Table 25 reports the per-layer speedup of `codeNx8` over each SOTA baseline for all configurations. Moreover, compared with AQLM's existing path for 3-5 codebooks, `codeNx8` is 2.25-$85.6\times$ faster.

For the Hadamard variant, DbyD (Had.+35W), we use `codeNx8_Had`, which applies the Hadamard transform to the input vector before building the lookup table; the reported latencies include this transform. Table 26 shows that the extra step reduces the speedups slightly, but `codeNx8_Had` is still faster than all baselines in all configurations, displayed in the main text with Figure 8.

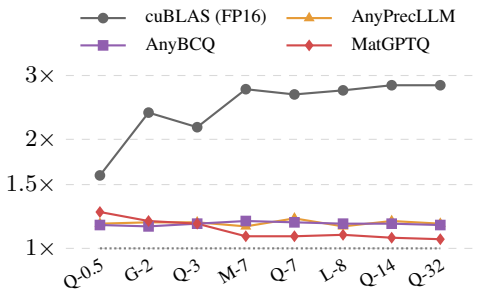


Figure 16: Latency speedup of `codeNx8` (↑). Values from Table 25

Table 25: Latency speedup of `codeNx8`. Values above 1.00× mean `codeNx8` is faster.

			vs. cuBLAS			vs. AnyBCQ			vs. AnyPrecisionLLM			vs. MatGPTQ	
Model	N	K	3-bit	4-bit	5-bit	3-bit	4-bit	5-bit	3-bit	4-bit	5-bit	3-bit	4-bit
Qwen2.5 0.5B	896	4864	1.79×	1.74×	1.59×	1.13×	1.19×	1.14×	1.21×	1.22×	1.26×	1.25×	1.23×
	4864	896	1.78×	1.72×	1.60×	1.15×	1.18×	1.17×	1.17×	1.09×	1.13×	1.26×	1.23×
	896	896	1.38×	1.37×	1.43×	1.14×	1.13×	1.18×	1.14×	1.12×	1.17×	1.29×	1.29×
Qwen2.5 3B	2048	11008	2.90×	2.53×	2.25×	1.21×	1.11×	1.12×	1.35×	1.26×	1.44×	1.28×	1.21×
	11008	2048	2.79×	2.47×	2.21×	1.22×	1.19×	1.16×	1.11×	1.04×	1.15×	1.06×	1.05×
	2048	2048	1.67×	1.60×	1.54×	1.16×	1.16×	1.17×	1.12×	1.07×	1.14×	1.25×	1.20×
Qwen2.5 7B	3584	18944	3.62×	2.95×	2.47×	1.16×	1.16×	1.16×	1.30×	1.20×	1.29×	1.10×	1.09×
	18944	3584	3.46×	2.84×	2.41×	1.24×	1.21×	1.21×	1.17×	1.11×	1.18×	1.03×	1.03×
	3584	3584	2.46×	2.19×	1.99×	1.14×	1.18×	1.14×	1.27×	1.12×	1.26×	1.13×	1.08×
Qwen2.5 14B	5120	13824	3.61×	2.93×	2.41×	1.24×	1.16×	1.13×	1.24×	1.17×	1.22×	1.11×	1.07×
	13824	5120	3.48×	2.86×	2.41×	1.20×	1.21×	1.20×	1.19×	1.14×	1.17×	1.05×	1.05×
	5120	5120	3.01×	2.62×	2.35×	1.18×	1.15×	1.10×	1.23×	1.12×	1.27×	1.09×	1.06×
Qwen2.5 32B	5120	27648	3.69×	2.91×	2.39×	1.17×	1.17×	1.13×	1.15×	1.12×	1.22×	1.08×	1.06×
	27648	5120	3.66×	2.88×	2.38×	1.25×	1.20×	1.17×	1.14×	1.10×	1.15×	1.05×	1.05×
	5120	5120	2.95×	2.58×	2.32×	1.11×	1.11×	1.11×	1.24×	1.12×	1.28×	1.08×	1.07×
Gemma 2B	2048	16384	3.33×	2.84×	2.46×	1.13×	1.12×	1.10×	1.32×	1.20×	1.37×	1.25×	1.16×
	16384	2048	3.30×	2.85×	2.49×	1.24×	1.13×	1.15×	1.13×	1.05×	1.15×	1.08×	1.05×
	2048	2048	1.68×	1.66×	1.54×	1.14×	1.21×	1.17×	1.15×	1.12×	1.14×	1.31×	1.30×
Mistral 7B-v0.3	4096	14336	3.61×	2.93×	2.46×	1.35×	1.15×	1.14×	1.24×	1.12×	1.21×	1.11×	1.08×
	14336	4096	3.52×	2.85×	2.45×	1.30×	1.18×	1.21×	1.16×	1.07×	1.12×	1.07×	1.05×
	4096	4096	2.65×	2.41×	2.18×	1.09×	1.13×	1.16×	1.16×	1.09×	1.19×	1.11×	1.09×
Llama-3 8B	4096	14336	3.62×	2.96×	2.47×	1.17×	1.17×	1.14×	1.22×	1.13×	1.22×	1.12×	1.08×
	14336	4096	3.51×	2.88×	2.42×	1.21×	1.22×	1.18×	1.15×	1.07×	1.10×	1.07×	1.05×
	4096	4096	2.61×	2.37×	2.12×	1.12×	1.20×	1.14×	1.17×	1.12×	1.20×	1.12×	1.09×

Table 26: Latency speedup of DbyD-Hadamard. Values above $1.00\times$ mean `codeNx8_Had` is faster.

			vs. cuBLAS			vs. AnyBCQ			vs. AnyPrecisionLLM			vs. MatGPTQ	
Model	N	K	3-bit	4-bit	5-bit	3-bit	4-bit	5-bit	3-bit	4-bit	5-bit	3-bit	4-bit
Qwen2.5 0.5B	896	4864	1.70×	1.66×	1.55×	1.11×	1.14×	1.13×	1.13×	1.19×	1.25×	1.18×	1.19×
	4864	896	1.63×	1.63×	1.55×	1.07×	1.15×	1.15×	1.08×	1.06×	1.07×	1.15×	1.16×
	896	896	1.28×	1.27×	1.28×	1.08×	1.08×	1.10×	1.09×	1.06×	1.08×	1.18×	1.17×
Qwen2.5 3B	2048	11008	2.87×	2.48×	2.22×	1.21×	1.10×	1.09×	1.35×	1.22×	1.46×	1.29×	1.17×
	11008	2048	2.71×	2.44×	2.18×	1.12×	1.14×	1.14×	1.09×	1.05×	1.12×	1.02×	1.04×
	2048	2048	1.59×	1.58×	1.50×	1.09×	1.14×	1.16×	1.09×	1.07×	1.14×	1.19×	1.16×
Qwen2.5 7B	3584	18944	3.58×	2.93×	2.46×	1.15×	1.15×	1.15×	1.29×	1.19×	1.29×	1.11×	1.06×
	18944	3584	3.42×	2.81×	2.38×	1.27×	1.19×	1.19×	1.16×	1.11×	1.17×	1.02×	1.02×
	3584	3584	2.37×	2.14×	1.95×	1.12×	1.13×	1.09×	1.25×	1.11×	1.23×	1.10×	1.07×
Qwen2.5 14B	5120	13824	3.53×	2.88×	2.42×	1.23×	1.14×	1.14×	1.22×	1.15×	1.23×	1.07×	1.05×
	13824	5120	3.47×	2.83×	2.39×	1.25×	1.20×	1.20×	1.19×	1.12×	1.16×	1.05×	1.04×
	5120	5120	2.86×	2.53×	2.27×	1.10×	1.14×	1.11×	1.20×	1.12×	1.27×	1.05×	1.04×
Qwen2.5 32B	5120	27648	3.65×	2.89×	2.37×	1.16×	1.16×	1.16×	1.14×	1.11×	1.21×	1.07×	1.06×
	27648	5120	3.63×	2.87×	2.36×	1.24×	1.19×	1.18×	1.13×	1.09×	1.14×	1.05×	1.04×
	5120	5120	2.87×	2.52×	2.26×	1.09×	1.13×	1.09×	1.19×	1.10×	1.26×	1.05×	1.04×
Gemma 2B	2048	16384	3.25×	2.82×	2.46×	1.12×	1.12×	1.20×	1.30×	1.18×	1.39×	1.21×	1.14×
	16384	2048	3.20×	2.76×	2.45×	1.21×	1.11×	1.17×	1.10×	1.04×	1.13×	1.04×	1.02×
	2048	2048	1.59×	1.58×	1.51×	1.09×	1.15×	1.15×	1.08×	1.08×	1.12×	1.18×	1.17×
Mistral 7B-v0.3	4096	14336	3.50×	2.92×	2.45×	1.28×	1.16×	1.15×	1.19×	1.12×	1.21×	1.07×	1.06×
	14336	4096	3.46×	2.85×	2.41×	1.26×	1.22×	1.18×	1.14×	1.07×	1.11×	1.05×	1.04×
	4096	4096	2.57×	2.33×	2.11×	1.10×	1.13×	1.10×	1.16×	1.08×	1.20×	1.09×	1.07×
Llama-3 8B	4096	14336	3.52×	2.91×	2.45×	1.15×	1.18×	1.15×	1.21×	1.12×	1.22×	1.08×	1.06×
	14336	4096	3.44×	2.82×	2.42×	1.26×	1.17×	1.20×	1.13×	1.06×	1.10×	1.04×	1.03×
	4096	4096	2.58×	2.34×	2.12×	1.10×	1.11×	1.09×	1.16×	1.09×	1.20×	1.10×	1.06×