ADAPTIVE MIXED-PRECISION CODEBOOK QUANTIZATION FOR LARGE LANGUAGE MODELS

Anonymous authors
Paper under double-blind review

ABSTRACT

Codebook quantization enables efficient compression of large language models to low bit-widths, making it suitable for resource-constrained deployment. Existing methods assign a uniform codebook configuration to all matrices. However, under a tight average-bit budget, this uniform configuration leads to capacity misallocation: insensitive linear maps receive disproportionate precision, while sensit ive projection layers remain under-represented. To address this issue, we propose Adaptive Mixed-Precision Codebook Quantization (AMPCQ), a framework that assigns different codebook configurations based on per-matrix sensitivity. The proposed method combines: (i) a heterogeneous tier space comprising single- and dual-codebook quantizers; (ii) a normalized activation-aware reconstruction metric for cross-matrix comparability; (iii) Pareto-filtered greedy allocation guided by marginal return per bit; and (iv) lightweight codebook-delta adaptation for downstream tasks. On Llama3-8B and Qwen3-14B, AMPCQ demonstrates superior performance over uniform AQLM-style allocation at matched average bit width. Specifically, when Qwen3-14B is compressed to an average bit width of approximately 3 bits, it retains 97.3% of its original average zero-shot accuracy. Our code is available in the Supplementary Material.

1 INTRODUCTION

Codebook quantization (Babenko & Lempitsky, 2014) is a well-established approach for extreme compression of Large Language Models (LLMs). Methods such as Additive Quantization of Language Models (AQLM) (Egiazarian et al., 2024) routinely outperform naive scalar rounding at matched storage budgets. However, existing codebook quantization methods adopt a uniform design: every weight matrix in the model uses the same number of codebooks, the same codebook sizes, and the same grouping configuration, thereby sharing an identical realized bit rate.

While uniform designs are straightforward to deploy, they are often incongruent with the heterogeneous nature of Transformers. Attention and Multi-Layer Perceptron (MLP) projections exhibit different sensitivity to quantization noise; depth and width vary significantly across layers. Figure 1(a) supports this point on Llama3-8B with a diagnostic additive codebook reconstruction at approximately 2 bits held fixed for every block. The subplot traces per-layer mean squared reconstruction error for one recurring linear map type, MLP *down* projections, so the *same* operator role is measured at increasing depth (log-scaled axis). The spread across layers reaches orders of magnitude (max-to-min ratios about 10^3–10^4 in our run), showing that quantization difficulty varies sharply with depth even when codebook capacity is nominally uniform. Consequently, under a tight model-level average-bit constraint, a fixed uniform tier tends to over-allocate codebook capacity to well-conditioned matrices while providing insufficient protection for sensitive ones. We refer to this failure mode as budget misallocation.

To address this budget misallocation, we propose Adaptive Mixed-Precision Codebook Quantization (AMPCQ). Each linear map picks one tier from a compact palette (1.0/1.5/2.0/3.0/4.0 nominal bits in our experiments). Lower tiers favor efficient single-codebook parameterizations. Higher tiers can activate dual additive codebooks when extra structured capacity is warranted.

Figure 1 summarizes the motivation behind AMPCQ. Panel (a) reveals pronounced heterogeneity in weight reconstruction error across both decoder depth and projection types under the same 2-bit codebook configuration. Here, MSE serves only as a diagnostic visualization of this heterogeneity;

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.

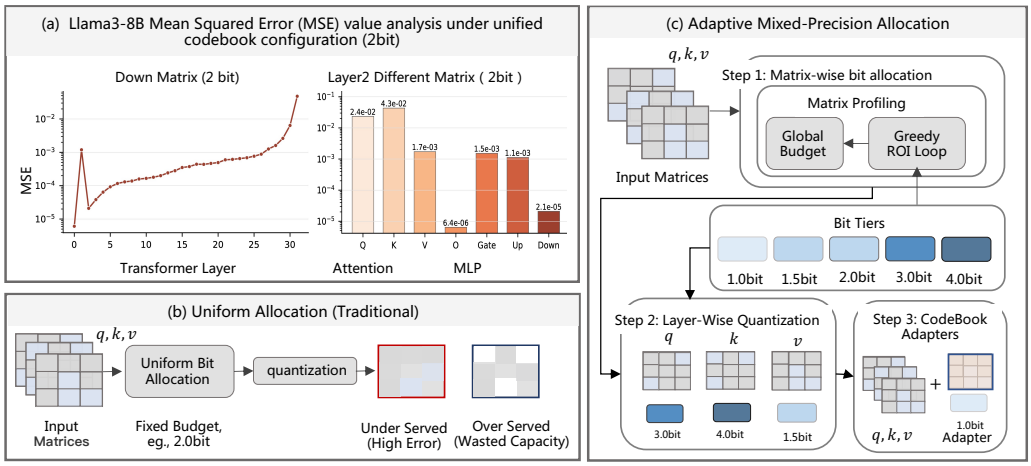

Figure 1: **Empirical heterogeneity and adaptive codebook quantization.** (a) Reconstruction MSE under a uniform 2-bit codebook configuration shows substantial variation across decoder layers and linear maps. (b) Uniform bit allocation can under-allocate difficult matrices while wasting capacity on easier ones. (c) AMPCQ profiles individual matrices and greedily assigns heterogeneous quantization tiers under a global average-bit budget, followed by layer-wise codebook fitting and lightweight codebook adapters.

Stage 1 instead uses the normalized activation-aware reconstruction loss in Eq. (4) to enable comparable sensitivity estimates across matrices. Panels (b) and (c) then illustrate how uniform allocation can misallocate capacity and how AMPCQ redistributes the global bit budget through matrix-wise tier assignment. Panel (c) contrasts this with two pipelines: uniform assignment locks every matrix to one tier and cannot steer bits toward the highest-error maps; AMPCQ keeps one model-level average-bit budget but lets each matrix take a possibly different tier, reallocating discrete codebook capacity according to profiling rather than a fixed structure.

Motivated by Figure 1(c), AMPCQ first profiles candidate tiers per matrix on calibration activations using a normalized reconstruction loss, Pareto-filters dominated rate–distortion operating points, and allocates the shared bit budget through greedy upgrades ranked by marginal distortion reduction per additional bit, which we term the return on investment (ROI). The second stage then assigns each linear map to a selected codebook tier; we fit additive codebooks (including dual-codebook configurations when selected) and scales, assign the corresponding indices, and perform layer-wise block recovery as in standard codebook post-training quantization (PTQ). An optional third stage trains lightweight task-specific codebook-delta adapters while freezing indices and base codebooks, preserving a compact backbone. Theoretically, we prove an equal-cost exchange inequality under uniform incremental bit costs. Marginal-return greedy budgeting is globally optimal only in the degenerate single-chain case ($n = 1$). For $n > 1$, the problem becomes a precedence-constrained knapsack (Kellerer et al., 2004), without guarantees of global optimality or approximation.

We evaluate Llama3-8B and Qwen3-14B under matched average bit width. The primary baselines are uniform codebook methods at the same bit width, such as AQLM (Egiazarian et al., 2024) and PocketLLM (Tian et al., 2026). Evaluation tasks include language modeling tasks and five zero-shot reasoning tasks. Compared to the competitive baseline AQLM at matched configuration, Llama3-8B at approximately 3-bit average bit width improves perplexity from 8.08/13.96 to 6.90/11.88 and increases mean accuracy from 63.27 to 67.16. On Qwen3-14B, at approximately 3-bit average bit width, the model retains about 97.3% of the dense model's aggregate score. Additionally, end-to-end throughput benchmarks for Llama-3-8B on NVIDIA A100 80GB and RTX 4090 GPUs show that our method achieves higher throughput than AQLM. Our contributions are summarized as follows:

- This work introduces a dynamic mixed-codebook quantization method based on the matrix structure of large language models, addressing the misallocation problem inherent in uniform codebook assignment.
- Extensive experiments demonstrate that AMPCQ consistently outperforms uniform codebook assignment at matched average bit width, achieving lower perplexity and higher mean zero-shot accuracy across both model families.

- Under standard Lipschitz assumptions, we prove that the worst-case forward error is at most linear in the layerwise Frobenius weight error, motivating a reconstruction-centric Stage 2 under mixed tiers.

2 Related Work and Background

Model compression for LLMs. LLM compression has traditionally included pruning, quantization, low-rank factorization, and knowledge distillation (Liu et al., 2025; Xu et al., 2024; Ilin & Richtarik, 2025; Lin et al., 2024; Wang et al., 2025; Sieberling et al., 2025). These approaches targeted different quality–efficiency trade-offs. Our work is closest to post-training quantization (PTQ), but is complementary to other compression directions. Targeting deployment under a fixed model-level bit budget, we allocate precision across matrices rather than imposing a uniform configuration.

Post-training quantization and codebook-based compression. PTQ was practical because it avoided full retraining. Generative Pre-trained Transformer Quantization (GPTQ) (Frantar et al., 2023) used Hessian-weighted reconstruction; Activation-aware Weight Quantization (AWQ) (Lin et al., 2024) used activation-aware importance. In the 2–4 bit regime, additive codebook methods were especially competitive; AQLM (Egiazarian et al., 2024) and PV-Tuning (Malinovskii et al., 2024) reported competitive accuracy under extreme compression. These pipelines usually applied *one* codebook configuration to every weight matrix; we study *matrix-wise* assignment from a small tier set that varies both codebook *structure* and *rate* under a single model-level average bit width.

Mixed-precision PTQ: scalar choices versus codebook tiers. Classical mixed-precision PTQ varied *scalar* bit-widths or numeric formats by layer or block sensitivity (Dong et al., 2019; Liu et al., 2026). A separate line of work utilized block reconstruction, budgeted greedy updates, and rate-distortion style simplifications in PTQ (Li et al., 2021; Yao et al., 2021; Nagel et al., 2021); the same algorithmic tools could apply when the per-layer candidate set is a set of *scalar* precisions. In codebook quantization, a candidate was not a bit-width alone: it was a tuple (K, S, g) that specified codebook count, size, and grouping, and it could switch between single- and dual-codebook parameterizations. AMPCQ applies Pareto filtering on empirical tier rate–distortion pairs $(R_i(\tau), D_i(\tau))$ followed by greedy budgeting ranked by a marginal-return (ROI) score (Li et al., 2021; Nagel et al., 2021); candidates are codebook tiers (K, S, g), not scalar precisions alone.

Efficient adaptation is important for compressed LLMs, especially in multi-task and edge use. Low-Rank Adaptation (LoRA)-style parameter-efficient fine-tuning (PEFT) (Hu et al., 2022) adds separate low-rank branches.

Overall, AMPCQ targets *joint* selection of codebook *structure* and *rate* per matrix; the next section makes the tier space and allocation objective explicit.

3 Method

AMPCQ compresses LLMs in a weight-only setting. Section 3.1 specifies the building block: each weight block is reconstructed by additive codebooks, and a *tier* chooses codebook count, codebook size, and block grouping, which together fix both the average bit rate and the single- vs. dual-codebook structure. **Stage 1** assigns each matrix one of five tiers under a global average-bit budget. **Stage 2** fits codebooks, scales, and discrete codes with layer-wise block recovery, reusing Stage 1's calibration activations. **Stage 3** trains small codebook-delta residuals for downstream tasks.

3.1 Codebook representation and tier space

Let $\mathcal{W} = \{W_i\}_{i=1}^{n}$ denote target linear matrices, with $W_i \in \mathbb{R}^{d_{\text{out}} \times d_{\text{in}}}$. For matrix i, we partition weights into blocks with output grouping size g_{out} and input grouping size g_{in}. Under tier τ, a block vector w is reconstructed by additive codebooks:

$$Q_\tau(w) = s \cdot \sum_{k=1}^{K} C_k[z_k], \qquad z_k \in \{1, \ldots, S\}, \tag{1}$$

where K is the number of codebooks, S is the codebook size per codebook, C_k is the k-th codebook, z_k is a discrete assignment, and s is a scale factor. A tier is defined as

$$\tau = (K, S, g_{\text{out}}, g_{\text{in}}). \tag{2}$$

Increasing K or S usually improves reconstruction quality, but it also increases storage and search cost. Using smaller groups increases representational flexibility, but it also introduces higher index overhead. Table 1 summarizes the concrete tier configurations used in this paper. It defines the parameter settings that correspond to each target bit level. In the low-bit regime, the candidate set is

$$\mathcal{T} = \{1.0\text{bit},\ 1.5\text{bit},\ 2.0\text{bit},\ 3.0\text{bit},\ 4.0\text{bit}\}. \tag{3}$$

The first three tiers use low-rate single-codebook forms, and the last two tiers use higher-capacity additive forms.

Tier grid design. We fix five nominal tiers (Table 1)—coarse enough for feasible Stage 1 allocation over experiments near 1–4 avg. bits, without neural architecture search (NAS) or exhaustive (K, S, g) search. Bit figures follow Appendix A.3; low tiers increase capacity via larger S and tighter g under $K{=}1$, while 3.0/4.0 bit switch to $K{=}2$ dual-codebook stacks (AQLM-style, i.e., a structural upgrade beyond scalar precision). Finer grids are future work (Section 6). The central design choice is matrix-wise tier selection. Two matrices, such as attention and MLP projections, can use different (K, S, g_{in}) values under the same model-level budget.

Table 1: Discrete tier palette for AMPCQ. K counts additive codebooks, S is codebook size, and $(g_{\text{out}}, g_{\text{in}})$ is per-block grouping; each row is one Stage 1 candidate and its Stage 2 quantizer shape.

Tier	K	S	$(g_{\text{out}}, g_{\text{in}})$	Encoding
1-bit	1	256	(1,8)	single codebook
1.5-bit	1	4096	(1,8)	single codebook
2-bit	1	65536	(1,8)	single codebook
3-bit	2	4096	(1,8)	dual codebook
4-bit	2	65536	(1,8)	dual codebook

3.2 Stage 1: Matrix-wise bit allocation

Stage 1 chooses a tier $\tau_i \in \mathcal{T}$ for each matrix W_i. Let X_i denote calibration activations at the input of W_i in the dense model, and let $Q_\tau(W_i)$ denote the quantized reconstruction under tier τ. We score each candidate tier by a normalized activation-aware reconstruction loss:

$$D_i(\tau) = \frac{\|X_i W_i^\top - X_i Q_\tau(W_i)^\top\|_F^2}{\|X_i W_i^\top\|_F^2 + \varepsilon}. \tag{4}$$

Here $D_i(\tau)$ measures how much the quantized linear map perturbs *outputs* on calibration inputs; dividing by $\|X_i W_i^\top\|_F^2$ makes values comparable across matrices of different scales. This is conceptually related to activation- and Hessian-weighted reconstruction in scalar PTQ (Frantar et al., 2023; Lin et al., 2024), but the goal here is *cross-matrix comparability* of $D_i(\tau)$ for tier ranking, not a new weight-space metric. For each tier τ, we pair distortion $D_i(\tau)$ with a rate $R_i(\tau)$, defined as the storage cost (bits) required by that tier for matrix i. The explicit storage decomposition for $R_i(\tau)$ and its equivalent average-bit form used by AMPCQ are given in Appendix A.3; $R_i(\tau)$ is not a single scalar bit-width but includes codebook and index terms tied to (K, S, g). This yields a trade-off over the *tier* set: lower $R_i(\tau)$ saves storage but typically increases $D_i(\tau)$.

Pareto filtering. Among tiers, some candidates are strictly worse than others on both axes (they cost more bits without reducing distortion, or reduce distortion without saving bits). This Pareto reduction of candidate precisions is standard in budgeted PTQ analyses (Li et al., 2021; Nagel et al., 2021); the distinction is that our points are codebook *tiers* $(R_i(\tau), D_i(\tau))$, not scalar bit levels. We discard such *dominated* points and keep only the *Pareto frontier*, i.e., the lowest-distortion achievable at each feasible rate for matrix i. Equivalently, we retain the lower envelope of the empirical rate–distortion curve.

Budgeted allocation. We allocate bits across matrices by minimizing weighted distortion subject to a weighted average-bit budget:

$$\min_{\{\tau_i\}} \sum_{i=1}^{n} \alpha_i D_i(\tau_i) \quad \text{s.t.} \quad \sum_{i=1}^{n} \alpha_i R_i(\tau_i) \le B,\ \tau_i \in \mathcal{T}, \tag{5}$$

where B is the target budget and $\alpha_i \geq 0$ weights matrices (for example by parameter count) so that frequent or large matrices contribute more to the objective and to the bit tally. We implement this budget by greedy upgrades, following the same high-level pattern as other discrete budgeted PTQ methods (Li et al., 2021; Yao et al., 2021). For an upgrade $\tau_a \to \tau_b$, let the distortion gain and extra rate be

$$G_i(\tau_a \to \tau_b) = D_i(\tau_a) - D_i(\tau_b), \qquad C_i(\tau_a \to \tau_b) = R_i(\tau_b) - R_i(\tau_a). \tag{6}$$

We prioritize upgrades using a marginal-return score (an ROI-style ratio):

$$\mathrm{ROI}_i(\tau_a \to \tau_b) = \frac{G_i(\tau_a \to \tau_b)}{C_i(\tau_a \to \tau_b)^\gamma}, \tag{7}$$

The exponent γ rescales the marginal bit cost C_i in the denominator. If $\gamma = 1$, then $\mathrm{ROI}_i = G_i/C_i$ is exactly the *marginal* distortion reduction per additional bit for the upgrade $\tau_a \to \tau_b$ (i.e., benefit per bit). The greedy procedure repeatedly picks the feasible upgrade with the largest ROI_i; with $\gamma = 1$ this is equivalent to ranking by marginal gain per bit. If $\gamma > 1$, the same score penalizes upgrades with larger C_i more strongly than the linear ($\gamma = 1$) case. In our implementation, the default is $\gamma = 1$, and all experiments in this paper use $\gamma = 1$ unless stated otherwise. Stage 1 outputs the resulting matrix-wise tier assignment $\{\tau_i\}_{i=1}^n$.

3.3 Stage 2: Layer-wise quantization

In this section, we refine the quantized parameters locally, one transformer block at a time ("layer-wise" here means one block F_l, not necessarily a single linear layer). Let Θ_l be the dense parameters inside block l and let $\widehat{\Theta}_l$ be their quantized counterparts. We collect inputs x_l from the dense model on the calibration set; x_l is the activation tensor entering block l. We then minimize the squared output mismatch between the dense reference block $F_l(\cdot; \Theta_l)$ and the quantized module $F_l(\cdot; \widehat{\Theta}_l)$:

$$\mathcal{L}_l = \mathbb{E}_{x_l} \| F_l(x_l; \Theta_l) - F_l(x_l; \widehat{\Theta}_l) \|_2^2, \tag{8}$$

Thus Eq. 8 targets alignment of block outputs on calibration inputs, which can differ from minimizing weight-space error alone.

Optimization splits into two coupled problems because $\widehat{\Theta}_l$ mixes continuous parameters (codebook vectors and scales) with discrete decisions (code indices). We therefore alternate updates: **(i)** gradient steps on the continuous parameters using $\mathcal{L}_l$; **(ii)** discrete search over code assignments, where candidate moves are judged by how much they reduce the *activation/output* error in Eq. 8 (activation-aware search), not only weight reconstruction error. Because Eq. 8 is optimized locally per block with dense activations as inputs, Stage 2 reduces block-level error without introducing a separate end-to-end objective over the full sequence.

3.4 Stage 3: Multi-task codebook-space adapters

For a downstream task t, we keep all shared codebooks and index maps frozen, and only introduce a small task-specific update. Let Q_i denote the effective quantized weight (or forward operator) of matrix i in that frozen backbone. We add a task residual $A_i^{(t)}$ that is trained in the *same* codebook parameterization as the backbone ("codebook space"), rather than as an unconstrained dense matrix:

$$Y_i^{(t)} = X_i \big(Q_i + A_i^{(t)}\big)^\top. \tag{9}$$

Concretely, $A_i^{(t)}$ is implemented with its own small codebooks, indices, and scales $(C^{(t)}_{\mathrm{adapter}}, Z^{(t)}_{\mathrm{adapter}}, S^{(t)}_{\mathrm{adapter}})$. Let $\mathrm{Dequant}(\cdot)$ map that discrete representation to a dense-style correction $\Delta W_i^{(t)}$ with the same shape as Q_i. Then

$$\Delta W_i^{(t)} = \mathrm{Dequant}\Big(C^{(t)}_{\mathrm{adapter}}, Z^{(t)}_{\mathrm{adapter}}, S^{(t)}_{\mathrm{adapter}}\Big), \tag{10}$$

and the forward pass can be written equivalently as

$$Y_i^{(t)} = X_i \left(Q_i + \Delta W_i^{(t)}\right)^\top. \tag{11}$$

Compared with parameter-efficient fine-tuning methods that insert separate low-rank pathways Hu et al. (2022), this design stays inside the codebook representation used by the backbone. Task switching swaps only the lightweight adapter parameters while the shared quantized weights remain unchanged, which limits interference across tasks.

4 THEORETICAL ANALYSIS

This section complements AMPCQ with two short analyses (formal statements in Appendix A). **(i)** A standard layerwise Lipschitz bound links Frobenius weight error to output error and motivates Stage 2 reconstruction; tier choice and profiling remain empirical (Section 3). **(ii)** After Stage 1 Pareto filtering, per-matrix upgrades form chains; we give an equal-cost exchange inequality (uniform incremental bit costs) and specify when marginal-return greedy budgeting is optimal versus only a heuristic when many chains interact.

4.1 APPROXIMATION ERROR PROPAGATION

Let F be a depth-L feed-forward network and $\widetilde{F}$ its quantized version. Assume each layer is Lipschitz in parameters and activations for inputs in a bounded set. Then the output gap $\|F(x;\theta) - \widetilde{F}(x;\widetilde{\theta})\|$ is upper-bounded by a linear form in the layerwise Frobenius norms $\|\Delta^{(\ell)}\|_F$ of weight errors $\Delta^{(\ell)}$. Small per-layer Frobenius quantization error therefore yields a proportionally small guaranteed output gap at this level of abstraction. Theorem A.1 in Appendix A states the closed-form bound. This argument does not determine which surrogate distortion best tracks end-to-end loss under mixed tiers. Stage 1 instead uses calibrated profiling (including activation normalization where we use it) as an engineering proxy, with empirical validation in Section 5.2.3. The proof expands the hidden-state gap at each depth into a parameter term and an activation term and unrolls the resulting recurrence.

4.2 GREEDY ALLOCATION UNDER CHAIN CONSTRAINTS

After Pareto filtering (Section 3.2), each matrix i keeps an ordered list of undominated tier upgrades with bit increments $\Delta b_{i,r}$ and distortion drops $\Delta e_{i,r}$. Any feasible plan satisfies *prefix* closure on each chain: taking upgrade r forces upgrades $1, \ldots, r-1$ on the same matrix. If $\Delta e_{i,r}/\Delta b_{i,r}$ is nonincreasing in r for every i, then ranking marginal upgrades by return per bit agrees with Eq. 7 at $\gamma{=}1$. Theorem A.2 (Appendix A) records an equal-cost exchange inequality under uniform incremental bit costs; it certifies a local improvement that raises objective value without changing total bit usage. The same theorem shows that the greedy frontier rule is globally optimal only in the single-chain case $n{=}1$ after filtering.

When $n{>}1$, coupled chains share one budget and define a discrete precedence-constrained allocation problem. Such knapsack variants are NP-hard in general (Kellerer et al., 2004). We do *not* claim global optimality or a constant-factor approximation for Stage 1's multi-chain greedy allocator.

5 EXPERIMENT

Our empirical study addresses four questions: **(Q1) Effectiveness at matched budgets.** At fixed average bit width (e.g., 2.0/3.0), does matrix-wise allocation outperform uniform codebook assignment and representative scalar PTQ methods in perplexity and zero-shot accuracy? **(Q2) Profiling and greedy ranking.** How much do (i) the per-matrix distortion metric $D_i(\tau)$ used to score candidate tiers and (ii) the incremental bit-cost term in the ROI-based upgrade order affect perplexity and mean zero-shot accuracy? **(Q3) Mixed tiers vs. uniform tier.** At matched average bit width, does heterogeneous matrix-wise tier assignment outperform forcing a *single* uniform tier on every matrix? **(Q4) Downstream behavior under compression.** After aggressive quantization, do models retain reasonable cross-domain utility on intent, summarization, and math tasks, and can lightweight adaptation further recover performance?

5.1 SETUP

5.1.1 MODELS AND COMPRESSION REGIMES

Our main experiments use Llama3-8B (AI@Meta, 2024) and Qwen3-14B (Team, 2025). We evaluate representative low-bit operating points corresponding to roughly 4–$8\times$ weight compression in our tables (2–4 avg. bits, depending on the row), and include dense references at 16 bits.

Table 2: Llama3-8B *weight-only* PTQ at equivalent average bit widths. AMPCQ utilizes Stage 1 allocation and Stage 2 recovery. **Avg_acc** denotes the mean accuracy across five zero-shot tasks.

Method	Avg_bits	Wiki2↓	C4↓	WinoGrande↑	PIQA↑	HellaSwag↑	ARC-E↑	ARC-C↑	Avg_acc↑
Llama3-8B	16	6.19	10.51	73.95	79.11	60.66	80.53	51.96	69.24
GPTQ	3.00	8.29	13.73	71.12	71.49	54.35	70.51	37.63	61.02
AWQ	3.00	11.4	16.4	72.10	77.61	55.13	74.01	43.18	64.41
RTN	3.00	27.9	110.28	54.71	62.29	29.07	32.10	22.53	40.14
QuIP#	3.00	9.51	14.28	72.53	76.80	26.63	72.91	41.02	63.70
AQLM	3.04	8.08	13.96	72.34	76.22	53.00	73.53	41.04	63.27
PocketLLM	2.98	12.53	15.00	**73.24**	76.99	57.23	74.37	46.08	65.58
AMPCQ	3.05	**6.90**	**11.88**	73.01	**78.45**	**58.08**	**78.57**	**47.70**	**67.16**
GPTQ	2.00	2.11E2	4.09E4	51.35	53.92	49.55	28.89	20.03	40.75
AWQ	2.00	1.71E6	1.97E6	51.10	55.94	26.37	25.29	21.31	36.00
RTN	2.02	1.90E3	2.43E4	51.19	53.02	25.58	24.79	21.60	35.24
QuIP#	2.00	85.13	1.25E2	51.70	52.93	29.28	29.01	21.28	36.84
AQLM	2.29	9.36	**12.46**	**71.95**	**77.75**	**55.40**	74.20	41.13	64.09
PocketLLM	2.15	12.67	18.60	51.70	63.28	32.95	47.35	23.46	52.36
AMPCQ	2.29	**8.22**	13.47	71.27	77.26	54.43	**76.09**	**44.45**	**64.89**

5.1.2 DATASETS AND EVALUATION METRICS

Language modeling. We report perplexity on WikiText-2 (Merity et al., 2016) and C4 (Raffel et al., 2020) under the standard next-token evaluation setup used in PTQ studies.

Zero-shot reasoning. We report accuracy on five widely used commonsense and reasoning benchmarks: WinoGrande (Sakaguchi et al., 2020), PIQA (Bisk et al., 2020), HellaSwag (Zellers et al., 2019), ARC-Easy, and ARC-Challenge (Clark et al., 2018). We aggregate them into the mean accuracy column **Avg_acc** reported in the tables. All zero-shot runs use the `lm-evaluation-harness` (Gao et al., 2024) with the same task configuration across methods.

Multi-task suite (supplementary evaluation). Beyond perplexity and zero-shot benchmarks, we additionally report a small multi-task benchmark (intent, summarization, and math) described in Section 5.2.2; dataset identifiers and metric definitions are summarized in Appendix B.

5.1.3 BASELINES

We compare against uniform AQLM-style codebook quantization at the *same* average bit budget (primary controlled baseline). We additionally include widely used scalar PTQ methods reported in our tables: GPTQ (Frantar et al., 2023), AWQ (Lin et al., 2024), Round-To-Nearest (RTN) (Nagel et al., 2021), QuIP# (Tseng et al., 2024), and PocketLLM (Tian et al., 2026). We align baseline implementations to reference settings where applicable and record replication details for ambiguous cases in Appendix B, together with default AMPCQ hyperparameters, hardware usage, and full reproducibility artifacts.

5.2 RESULTS AND ANALYSIS

5.2.1 COMPRESSION QUALITY

We measure perplexity on WikiText-2 (Merity et al., 2016) and C4 (Raffel et al., 2020) and zero-shot accuracy on WinoGrande (Sakaguchi et al., 2020), PIQA (Bisk et al., 2020), HellaSwag (Zellers et al., 2019), ARC-Easy, and ARC-Challenge (Clark et al., 2018) via lm-eval-harness (Gao et al., 2024); protocol details appear in Section 5.1 and Appendix B. Tables 2 and 3 summarize the main comparisons.

Table 2 addresses **(Q1)** at $\sim 5\times$ (approximately 3-bit) and $\sim 8\times$ (approximately 2.3-bit). Relative to uniform AQLM at matched average bit width in each block, AMPCQ improves WikiText-2/C4 perplexity and mean zero-shot **Avg_acc**: $8.08/13.96 \rightarrow 6.90/11.88$ with $63.27 \rightarrow 67.16$ at $\sim 5\times$, and $9.36/12.46 \rightarrow 8.22/13.47$ with $64.09 \rightarrow 64.89$ on **Avg_acc** at $\sim 8\times$, where C4 is slightly worse than AQLM (13.47 vs. 12.46). Against scalar PTQ, mean accuracy spreads widely (GPTQ 61.02 vs. AMPCQ 67.16; PocketLLM 65.58). At $\sim 8\times$, many scalar runs blow up on perplexity whereas codebook models remain usable, and AMPCQ still improves Wiki2 and **Avg_acc** over AQLM.

Table 3: Qwen3-14B *weight-only* PTQ with the same metrics as Table 2. Upper block: average bit width of approximately 4 bits; lower block: average bit width of approximately 3 bits.

Method	Avg_bits	Wiki2↓	C4↓	WinoGrande↑	PIQA↑	HellaSwag↑	ARC-E↑	ARC-C↑	Avg_acc↑
Qwen3-14B	16	6.38	14.20	72.93	80.25	61.03	84.18	58.79	71.43
RTN	4.00	14.01	23.12	63.54	74.81	51.85	67.42	37.71	59.07
AWQ	4.00	**7.00**	15.23	65.40	76.30	53.50	76.40	51.00	64.52
GPTQ	4.00	7.11	14.96	73.20	**79.70**	**60.60**	81.50	54.00	69.80
PocketLLM	3.98	11.24	17.72	61.38	60.64	56.16	80.45	53.97	62.52
AQLM	3.94	8.75	15.03	70.17	79.11	54.57	78.87	55.20	67.58
AMPCQ	3.80	8.15	**14.56**	**73.40**	79.33	60.02	**83.67**	**58.11**	**70.90**
RTN	3.00	5.3E6	4.62E6	50.50	53.91	25.69	24.12	21.48	35.14
AWQ	3.00	1.26E7	1.18E7	50.10	53.31	25.31	24.71	23.19	35.32
GPTQ	3.00	5.68E3	1.12E3	51.01	51.09	24.80	26.01	23.50	35.28
PocketLLM	3.00	4.0E4	4.9E4	49.17	53.37	26.04	26.48	22.18	35.45
AQLM	3.04	9.26	16.35	70.15	77.69	54.47	79.92	50.01	66.45
AMPCQ	3.00	**8.61**	**15.20**	**72.37**	**78.51**	**58.64**	**82.53**	**55.63**	**69.53**

Table 3 reports the same metrics at $\sim 4\times$ (approximately 4-bit) and $\sim 5\times$ (approximately 3-bit). Against AQLM at matched average bit width, AMPCQ reduces WikiText-2/C4 perplexity from $8.75/15.03$ to $8.15/14.56$ at $\sim 4\times$ and from $9.26/16.35$ to $8.61/15.20$ at $\sim 5\times$, while lifting **Avg_acc** from 67.58 to 70.90 and from 66.45 to 69.53. Among other methods, GPTQ reaches 69.80 mean accuracy at $\sim 4\times$ but stays below AMPCQ (70.90). At $\sim 5\times$, AWQ, GPTQ, and RTN show catastrophic perplexity, whereas AMPCQ remains stable and achieves the strongest aggregate accuracy among usable entries.

5.2.2 MULTI-TASK EXPERIMENTS

We further evaluate post-quantization adaptation on Llama-3-8B using GSM8K (Cobbe et al., 2021), SAMSum (Gliwa et al., 2019), and BlendX (Yoon et al., 2024). We compare LoRA ($r = 36$) with the native codebook adapter under comparable trainable parameter budgets (See Table 7).

With LoRA, AMPCQ substantially improves GSM8K accuracy over AQLM (40.13% vs. 25.70%), while remaining comparable on SAMSum and BlendX. With native adapters, AMPCQ consistently outperforms AQLM across all three tasks, reaching 30.78% on GSM8K, 0.3298 ROUGE-L on SAMSum, and 0.4421 F1-macro on BlendX. These results show that AMPCQ supports effective downstream adaptation under a small trainable parameter budget, while the native codebook adapter provides a lightweight alternative to LoRA.

5.2.3 ABLATION STUDIES

This section varies Stage 1 tier assignment and profiling choices while holding calibration data and the codebook fitting protocol fixed, and reports WikiText-2/C4 perplexity plus mean accuracy over WinoGrande, PIQA, HellaSwag, ARC-Easy, and ARC-Challenge. To isolate the effect of Stage 1 allocation, we disable Stage 2 layer-wise recovery in these ablations unless otherwise specified. We also include a single-layer MLP latency check on an edge-class graphics processing unit (GPU) under single-token decoding.

Table 4: **Stage 1 ablation** (Llama3-8B): one uniform tier on every matrix vs. mixed matrix-wise tiers.

Method	Avg_bits	Wiki2↓	C4↓	Avg↑
Uniform	2.27	169.65	273.99	38.64
Mixed	2.27	**15.39**	**28.61**	**58.64**
Uniform	3.04	9.27	16.37	60.81
Mixed	3.04	**7.78**	**13.80**	**64.61**

(Q3) Mixed matrix-wise tiers vs. uniform tiers. Table 4 compares one forced uniform tier against mixed matrix-wise tiers at the same **Avg_bits** before Stage 2 recovery. At 2.27 bits, uniform rows report WikiText-2/C4 perplexity $169.65/273.99$ and mean accuracy 38.64; mixed rows report $15.39/28.61$ and 58.64. At 3.04 bits, mixed tiers yield lower perplexity ($7.78/13.80$ vs. $9.27/16.37$) and higher mean accuracy (64.61 vs. 60.81). The resulting allocation is non-uniform across both depth and projection types; we provide a qualitative visualization of these learned tier maps in Appendix Figure 3.

Table 6: End-to-end throughput (tok/s) and GPU peak memory on Llama-3-8B (2048-token context + 512-token generation).

Model	A100 80GB			RTX 4090 24GB		
	Batch	tok/s	GPU Peak	Batch	tok/s	GPU Peak
QuIP#-2bit (CUDA Graph)	1	77.8	6.5 GiB	1	35.8	6.5 GiB
AQLM-2bit	1	18.3	4.15 GiB	1	25.3	4.15 GiB
AMPCQ (Ours)	1	20.1	4.60 GiB	1	28.9	4.58 GiB
QuIP#-2bit (CUDA Graph)	26	182.3	75.9 GiB	4	99.2	19.84 GiB
AQLM-2bit	48	163.1	72.1 GiB	5	158.9	23.5 GiB
AMPCQ (Ours)	40	123.5	76.49 GiB	4	168.7	22.05 GiB

(Q2) Distortion metric for candidate scoring. Table 5 swaps the distortion $D_i(\tau)$ used to rank tier candidates while fixing the greedy budget pipeline and Stage 2. Activation normalized mean squared error (NMSE) reaches $7.78/13.80$ perplexity and 64.61 mean accuracy in this grid; plain MSE/NMSE and activation MSE score higher perplexity or lower mean accuracy on the same axes. Activation MSE without normalization yields $2033.67/1320.64$ perplexity and 36.62 mean accuracy, indicating large sensitivity to how D_i is scaled across matrices.

Table 5: **Stage 1 ablation** (Llama3-8B): choice of per-matrix distortion $D_i(\tau)$ for ranking tier candidates (Section 3.2). Fixed greedy budget pipeline and codebook fitting protocol.

Metric	Wiki2↓	C4↓	Avg↑
Plain MSE	9.81	16.17	61.23
Plain NMSE	9.15	15.95	61.95
Activation MSE	2033.67	1320.64	36.62
Activation NMSE	**7.78**	**13.80**	**64.61**

Bit-aware ROI and robustness. Appendix Table 8 compares ROI ranking with and without bit-cost denominators (Eq. 7): including bit cost yields lower perplexity and higher mean accuracy at both 2.27 and 3.04 bits in that table. Appendix Figure 2 plots WikiText-2 perplexity versus γ at 2.27 and 3.04 bits; Appendix Table 9 varies profiling sample count (64–256) at 3.04 bits. Appendix Section C repeats analogous Stage 1 checks on Qwen3-14B under the same protocol.

End-to-End Throughput We measure end-to-end throughput on Llama-3-8B with a 2048-token context and 512-token generation on NVIDIA A100 80GB and RTX 4090 GPUs. To ensure a fair comparison, all methods are benchmarked with CUDA Graph support enabled via `torch.compile(max-autotune, fullgraph=True)`. Under the same optimized setting, our AMPCQ achieves competitive throughput across different batch sizes and GPU platforms, reaching 20.1 tok/s at batch-1 on A100 and 168.7 tok/s at batch-4 on RTX 4090, while maintaining lower memory overhead in several configurations. Appendix Section B.9 repeats the edge-device inference latency versus FP16.

6 Conclusion and Future Work

We introduced *Adaptive Mixed-Precision Codebook Quantization* (AMPCQ), which co-allocates codebook structure and rate per weight matrix under a model-level average-bit budget for extreme codebook LLM compression. Its pipeline (normalized output error profiling, Pareto tier pruning, marginal-return greedy scheduling) builds on budgeted scalar PTQ (Li et al., 2021; Yao et al., 2021; Nagel et al., 2021), with the tier rate–distortion surface as the central object rather than a new optimizer. Stage 2 fits codebook parameters via layer-wise recovery; Stage 3 documents codebook-delta fine-tuning for downstream task adaptation (Section 3.4). On Llama3-8B and Qwen3-14B, AMPCQ improves language modeling and mean zero-shot accuracy over uniform AQLM at matched average bit width (Tables 2 and 3). Section 5.2.3 and Appendix Sections C analyze Stage 1 sensitivity to tier mixing, distortion metrics, ROI weighting, and profiling sample count.

Future work could explore several promising directions, including finer-grained allocation (e.g., sub-matrix or expert blocks), hardware-aware mapping of heterogeneous tiers, runtime- or workload-adaptive budgets, and a rigorous comparison of codebook-delta adaptation against low-rank adapters under matched overhead.

AI USE STATEMENT

In this work, we used generative AI tools to aid and polish writing. Specifically, they were used to draft and restructure manuscript text, format LaTeX and tables, generate schematic figures from author-specified layouts, and search and format bibliographic metadata. We did not use generative AI tools to generate synthetic datasets, invent experimental numbers, or fabricate citations. Proposing hypotheses, implementing training code, and collecting metrics were done in the project codebase. We checked citations against arXiv, CVF Open Access, APS, and publisher pages. We reviewed all AI-assisted work and take responsibility for the final content of this work, including text, claims, and figures produced with the aid of generative AI.

ETHICS STATEMENT

The authors have read and agree to adhere to the ICLR Code of Ethics. Our experiments use only public datasets and pretrained models, and involve neither human subjects nor personal data. Our method is a quantization technique for LLM inference; beyond the risks inherent in the base models, we do not believe it introduces new ethical risks.

REPRODUCIBILITY STATEMENT

The supplementary material provides an anonymized code repository, configuration files, and evaluation scripts. All datasets and pretrained models used in this work are publicly available. Section 5.1 describes data processing, calibration, hyperparameter settings, and hardware and software versions. The main paper and appendix present the full AMPCQ pipeline, including tier construction, greedy scheduling, and Stage 2 and Stage 3 training details. The ablation settings and profiling configurations are given in Section 5.2.3 and Appendix Sections B.4 through C. The results in Tables 2, 3, and 6 can be reproduced with the provided scripts.

REFERENCES

AI@Meta. Llama 3 model card. 2024. URL https://github.com/meta-llama/llama3/blob/main/MODEL_CARD.md.

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

Yonatan Bisk, Rowan Zellers, Ronan Le Bras, Jianfeng Gao, and Yejin Choi. PIQA: Reasoning about physical commonsense in natural language. In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 34, pp. 7432–7439, 2020. URL https://ojs.aaai.org/index.php/AAAI/article/view/6239.

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. URL https://arxiv.org/abs/1803.05457.

Karl Cobbe, Vineet Kosaraju, Mohammad Bavarian, Mark Chen, Heewoo Jun, Lukasz Kaiser, Matthias Plappert, Jerry Tworek, Jacob Hilton, Reiichiro Nakano, Christopher Hesse, and John Schulman. Training verifiers to solve math word problems. *arXiv preprint arXiv:2110.14168*, 2021. URL https://arxiv.org/abs/2110.14168.

Zhen Dong, Zhewei Yao, Amir Gholami, Michael Mahoney, and Kurt Keutzer. Hawq: Hessian aware quantization of neural networks with mixed-precision. In *IEEE/CVF International Conference on Computer Vision*, pp. 293–302, 2019.

Vage Egiazarian, Andrei Panferov, Denis Kuznedelev, Elias Frantar, Artem Babenko, and Dan Alistarh. Extreme compression of large language models via additive quantization. In *Forty-first International Conference on Machine Learning*, 2024. URL https://openreview.net/forum?id=5mCaITRTmO.

Elias Frantar, Saleh Ashkboos, Torsten Hoefler, and Dan Alistarh. Gptq: Accurate post-training quantization for generative pre-trained transformers. In *International Conference on Learning Representations*, 2023. URL https://openreview.net/forum?id=ksVGCOlOEba.

Leo Gao, Jonathan Tow, Baber Abbasi, Stella Biderman, Sid Black, Anthony DiPofi, Charles Foster, Laurence Golding, Jeffrey Hsu, Alain Le Noac'h, Haonan Li, Kyle McDonell, Niklas Muennighoff, Chris Ociepa, Jason Phang, Laria Reynolds, Hailey Schoelkopf, Aviya Skowron, Lintang Sutawika, Eric Tang, Anish Thite, Ben Wang, Kevin Wang, and Andy Zou. The language model evaluation harness, 07 2024. URL https://zenodo.org/records/12608602.

Bogdan Gliwa, Iwona Mochol, Maciej Biesek, and Aleksander Wawer. SAMSum corpus: A human-annotated dialogue dataset for abstractive summarization. In *Proceedings of the 2nd Workshop on New Frontiers in Summarization*, pp. 70–79. Association for Computational Linguistics, 2019. doi: 10.18653/v1/D19-5409.

Edward J. Hu, Yelong Shen, Phillip Wallis, Zeyuan Allen-Zhu, Yuanzhi Li, Shean Wang, Lu Wang, and Weizhu Chen. Lora: Low-rank adaptation of large language models. In *International Conference on Learning Representations*, 2022. URL https://openreview.net/forum?id=nZeVKeeFYf9.

Wei Huang, Haotong Qin, Yangdong Liu, Yawei Li, Qinshuo Liu, Xianglong Liu, Luca Benini, Michele Magno, Shiming Zhang, and Xiaojuan Qi. SliM-LLM: Salience-driven mixed-precision quantization for large language models. In *Proceedings of the 42nd International Conference on Machine Learning*, volume 267 of *Proceedings of Machine Learning Research*, pp. 25672–25692. PMLR, 2025. URL https://proceedings.mlr.press/v267/huang25aa.html.

Ivan Ilin and Peter Richtarik. Thanos: A block-wise pruning algorithm for efficient large language model compression, 2025. URL https://arxiv.org/abs/2504.05346.

Hans Kellerer, Ulrich Pferschy, and David Pisinger. *Knapsack Problems*. Springer, Berlin, 2004.

Sehoon Kim, Coleman Richard Charles Hooper, Amir Gholami, Zhen Dong, Xiuyu Li, Sheng Shen, Michael W. Mahoney, and Kurt Keutzer. SqueezeLLM: Dense-and-sparse quantization. In *Proceedings of the 41st International Conference on Machine Learning*, volume 235 of *Proceedings of Machine Learning Research*, pp. 23901–23923. PMLR, 2024. URL https://proceedings.mlr.press/v235/kim24f.html.

Deokjae Lee and Hyun Oh Song. Q-Palette: Fractional-bit quantizers toward optimal bit allocation for efficient LLM deployment. In *Advances in Neural Information Processing Systems*, volume 38, 2025. doi: 10.52202/085713-0387. URL https://proceedings.neurips.cc/paper_files/paper/2025/hash/10df8641edc837ca49360d5fb1d182f0-Abstract-Conference.html.

Shiyao Li, Xuefei Ning, Ke Hong, Tengxuan Liu, Luning Wang, Xiuhong Li, Kai Zhong, Guohao Dai, Huazhong Yang, and Yu Wang. LLM-MQ: Mixed-precision quantization for efficient LLM deployment. In *The Efficient Natural Language and Speech Processing Workshop at NeurIPS*, 2023. URL https://neurips2023-enlsp.github.io/papers/paper_4.pdf.

Yuhang Li, Ruihao Gong, Xu Tan, Yang Yang, Peng Hu, Qi Zhang, Fengwei Yu, Wei Wang, and Shi Gu. BRECQ: Pushing the limit of post-training quantization by block reconstruction. In *International Conference on Learning Representations*, 2021. URL https://openreview.net/forum?id=POWv6hDd9XH.

Ji Lin, Jiaming Tang, Haotian Tang, Shang Yang, Wei-Ming Chen, Wei-Chen Wang, Guangxuan Xiao, Xingyu Dang, Chuang Gan, and Song Han. AWQ: Activation-aware weight quantization for on-device LLM compression and acceleration. In *Proceedings of the Seventh Annual Conference on Machine Learning and Systems*, 2024. URL https://proceedings.mlsys.org/paper_files/paper/2024/hash/42a452cbafa9dd64e9ba4aa95cc1ef21-Abstract-Conference.html.

Defu Liu, Yixiao Zhu, Zhe Liu, Yi Liu, Changlin Han, Jinkai Tian, Ruihao Li, and Wei Yi. A survey of model compression techniques: past, present, and future. *Frontiers in Robotics and AI*, 2025.

Wenyuan Liu, Haoqian Meng, Yilun Luo, Peng Zhang, and Xindian Ma. Micromix: Efficient mixed-precision quantization with microscaling formats for large language models. In *The Fourteenth International Conference on Learning Representations*, 2026. URL https://openreview.net/forum?id=P5OKoZdwlB.

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

Stephen Merity, Caiming Xiong, James Bradbury, and Richard Socher. Pointer sentinel mixture models, 2016. URL https://arxiv.org/abs/1609.07843.

Markus Nagel, Marios Fournarakis, Rana Anna Amjad, Yelysei Bondarenko, Tijmen van Baalen, and Taco Blankevoort. A white paper on neural network quantization. *arXiv preprint arXiv:2106.08295*, 2021.

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*, 21:140:1–140:67, 2020. URL https://jmlr.org/papers/v21/20-074.html.

Keisuke Sakaguchi, Ronan Le Bras, Chandra Bhagavatula, and Yejin Choi. Winogrande: An adversarial winograd schema challenge at scale. *Proceedings of the AAAI Conference on Artificial Intelligence*, 34(5):8732–8740, 2020.

Oliver Sieberling, Denis Kuznedelev, Eldar Kurtic, and Dan Alistarh. Evopress: Accurate dynamic model compression via evolutionary search. In *Proceedings of the 42nd International Conference on Machine Learning*, 2025.

Qwen Team. Qwen3 technical report, 2025. URL https://arxiv.org/abs/2505.09388.

Ye Tian, Chengcheng Wang, Jing Han, Yehui Tang, and Kai Han. PocketLLM: Ultimate compression of large language models via meta networks. In *Proceedings of the AAAI Conference on Artificial Intelligence*, pp. 33250–33258, 2026. doi: 10.1609/aaai.v40i39.40610.

Albert Tseng, Jerry Chee, Qingyao Sun, Volodymyr Kuleshov, and Christopher De Sa. Quip#: Even better LLM quantization with hadamard incoherence and lattice codebooks. In Ruslan Salakhutdinov, Zico Kolter, Katherine A. Heller, Adrian Weller, Nuria Oliver, Jonathan Scarlett, and Felix Berkenkamp (eds.), *Forty-first International Conference on Machine Learning, ICML 2024, Vienna, Austria, July 21-27, 2024*, volume 235 of *Proceedings of Machine Learning Research*, pp. 48630–48656. PMLR / OpenReview.net, 2024. URL https://proceedings.mlr.press/v235/tseng24a.html.

Jingcun Wang, Yu-Guang Chen, Ing-Chao Lin, Bing Li, and Grace Li Zhang. Basis sharing: Cross-layer parameter sharing for large language model compression. In *International Conference on Learning Representations*, 2025. URL https://openreview.net/forum?id=gp32jvUquq.

Peng Xu, Wenqi Shao, Mengzhao Chen, Shitao Tang, Kaipeng Zhang, Peng Gao, Fengwei An, Yu Qiao, and Ping Luo. BESA: Pruning large language models with blockwise parameter-efficient sparsity allocation. In *International Conference on Learning Representations*, 2024. URL https://openreview.net/forum?id=gC6JTEU3jl.

Zhewei Yao, Zhen Dong, Zhangcheng Zheng, Amir Gholami, Jiali Yu, Eric Tan, Leyuan Wang, Qijing Huang, Yida Wang, Michael W. Mahoney, and Kurt Keutzer. HAWQ-V3: Dyadic neural network quantization. In *Proceedings of the 38th International Conference on Machine Learning*, volume 139 of *Proceedings of Machine Learning Research*, pp. 11875–11886, 2021. URL http://proceedings.mlr.press/v139/yao21a.html.

Yejin Yoon, Jungyeon Lee, Kangsan Kim, Chanhee Park, and Taeuk Kim. BlendX: Complex multi-intent detection with blended patterns. In Nicoletta Calzolari, Min-Yen Kan, Veronique Hoste, Alessandro Lenci, Sakriani Sakti, and Nianwen Xue (eds.), *Proceedings of the 2024 Joint International Conference on Computational Linguistics, Language Resources and Evaluation (LREC-COLING 2024)*, pp. 2428–2439, Torino, Italia, May 2024. ELRA and ICCL. URL https://aclanthology.org/2024.lrec-main.218.

Rowan Zellers, Ari Holtzman, Yonatan Bisk, Ali Farhadi, and Yejin Choi. HellaSwag: Can a machine really finish your sentence? In *Proceedings of the Annual Meeting of the Association for Computational Linguistics*, pp. 4791–4800, 2019. doi: 10.18653/v1/P19-1472. URL https://aclanthology.org/P19-1472/.

A PROOFS FOR THEORETICAL ANALYSIS

This appendix provides formal results and proofs referenced from Section 4. We provide a layerwise Lipschitz propagation bound for theoretical context and consistent notation; this result serves as a baseline analysis rather than a primary rationale for activation-normalized profiling or for dual- versus single-codebook tiers. The second records an equal-cost exchange inequality for chain-structured tier upgrades and the case $n{=}1$, where greedy budgeting is optimal; for $n > 1$, the Stage 1 allocator is treated as a practical heuristic (Section 4). Layers are indexed by $\ell = 1, \dots, L$ along a depth-L feed-forward stacking. Each $W^{(\ell)}$ may bundle several linear maps inside one transformer block provided the composed layer map satisfies Assumption A.1.

A.1 PROPAGATION OF LAYERWISE QUANTIZATION ERROR

Assumption A.1 (Layer Lipschitz regularity) *For each layer ℓ, there exist constants $L_{\mathrm{par}}^{(\ell)}$ and $L_{\mathrm{act}}^{(\ell)}$ such that for all $h, \widetilde{h}$ in a bounded set $\mathcal{H}$ and all admissible parameter matrices $W^{(\ell)}, \widetilde{W}^{(\ell)}$,*

$$\left\| f^{(\ell)}(h; W^{(\ell)}) - f^{(\ell)}(h; \widetilde{W}^{(\ell)}) \right\| \le L_{\mathrm{par}}^{(\ell)} \left\| W^{(\ell)} - \widetilde{W}^{(\ell)} \right\|_F, \tag{12}$$

$$\left\| f^{(\ell)}(h; W^{(\ell)}) - f^{(\ell)}(\widetilde{h}; W^{(\ell)}) \right\| \le L_{\mathrm{act}}^{(\ell)} \left\| h - \widetilde{h} \right\|. \tag{13}$$

Assumption A.1 holds, for example, for affine maps followed by activation functions that are Lipschitz on $\mathcal{H}$, and more generally for compositions that are Lipschitz in parameters and inputs on bounded domains. Let $\Delta^{(\ell)} := W^{(\ell)} - \widetilde{W}^{(\ell)}$ denote the quantization error at layer ℓ.

Theorem A.1 (Layerwise Lipschitz propagation bound) *Under Assumption A.1, for any input x that produces hidden states in $\mathcal{H}$, the outputs of F and $\widetilde{F}$ satisfy*

$$\left\| F(x;\theta) - \widetilde{F}(x;\widetilde{\theta}) \right\| \le \sum_{\ell=1}^{L} \Big(L_{\mathrm{par}}^{(\ell)} \cdot \left\| \Delta^{(\ell)} \right\|_F \cdot \prod_{j=\ell+1}^{L} L_{\mathrm{act}}^{(j)} \Big). \tag{14}$$

In particular, if $\|\Delta^{(\ell)}\|_F \le \varepsilon^{(\ell)}$ for each ℓ, then the same bound holds with $\|\Delta^{(\ell)}\|_F$ replaced by $\varepsilon^{(\ell)}$. Equivalently, there exists a constant C depending only on $\{L_{\mathrm{act}}^{(j)}\}$, $\{L_{\mathrm{par}}^{(\ell)}\}$, and L such that $\sup_x \|F(x;\theta) - \widetilde{F}(x;\widetilde{\theta})\| \le C \sum_{\ell=1}^{L} \varepsilon^{(\ell)}$.

Proof. Write $h^{(0)} = \widetilde{h}^{(0)} = x$ and, for $\ell \ge 1$, $h^{(\ell)} = f^{(\ell)}(h^{(\ell-1)}; W^{(\ell)})$ and $\widetilde{h}^{(\ell)} = f^{(\ell)}(\widetilde{h}^{(\ell-1)}; \widetilde{W}^{(\ell)})$. For $\ell \ge 1$, the triangle inequality gives

$$\left\| h^{(\ell)} - \widetilde{h}^{(\ell)} \right\| \le \left\| f^{(\ell)}(h^{(\ell-1)}; W^{(\ell)}) - f^{(\ell)}(h^{(\ell-1)}; \widetilde{W}^{(\ell)}) \right\| \tag{15}$$

$$+ \left\| f^{(\ell)}(h^{(\ell-1)}; \widetilde{W}^{(\ell)}) - f^{(\ell)}(\widetilde{h}^{(\ell-1)}; \widetilde{W}^{(\ell)}) \right\|. \tag{16}$$

The first term is bounded by $L_{\mathrm{par}}^{(\ell)} \|\Delta^{(\ell)}\|_F$ and the second by $L_{\mathrm{act}}^{(\ell)} \|h^{(\ell-1)} - \widetilde{h}^{(\ell-1)}\|$. Define $e^{(\ell)} := \|h^{(\ell)} - \widetilde{h}^{(\ell)}\|$. Then for $\ell \ge 1$,

$$e^{(\ell)} \le L_{\mathrm{par}}^{(\ell)} \|\Delta^{(\ell)}\|_F + L_{\mathrm{act}}^{(\ell)} e^{(\ell-1)}, \qquad e^{(0)} = 0. \tag{17}$$

Unrolling Eq. 17 yields

$$e^{(L)} \le \sum_{\ell=1}^{L} L_{\mathrm{par}}^{(\ell)} \|\Delta^{(\ell)}\|_F \prod_{j=\ell+1}^{L} L_{\mathrm{act}}^{(j)}. \tag{18}$$

Since $F(x;\theta) = h^{(L)}$ and $\widetilde{F}(x;\widetilde{\theta}) = \widetilde{h}^{(L)}$, we obtain Eq. 14. Substituting $\|\Delta^{(\ell)}\|_F \le \varepsilon^{(\ell)}$ yields the first corollary inequality. The final bound of the form $C \sum_\ell \varepsilon^{(\ell)}$ follows because each product $\prod_{j=\ell+1}^{L} L_{\mathrm{act}}^{(j)}$ is bounded by $\big(\max_j \max(1, L_{\mathrm{act}}^{(j)})\big)^L$ uniformly in ℓ. □

Remark (Implications for AMPCQ). Theorem A.1 demonstrates that small Frobenius weight errors contribute linearly (with depth-dependent coefficients) to output error under Assumption A.1. It provides no prescription for whether Stage 1 should rank tiers using activation-normalized distortion versus plain Frobenius squared error, nor does it favor dual-codebook tiers at higher nominal rates: those choices affect the empirical maps $\tau \mapsto D_i(\tau)$ and appear only indirectly in any bound through realized $\|\Delta^{(\ell)}\|_F$. End-to-end tier selection therefore remains an empirical question, addressed by profiling and by the experiments in Section 5.2.3.

A.2 Greedy budgeting on chains: exchange inequality and the n=1 case

Definition 1 (Feasible upgrades after Pareto filtering) *Fix a matrix index $i \in \{1, \dots, n\}$. After Pareto filtering, suppose matrix i admits a finite ordered chain of tier upgrades $r = 1, \dots, K_i$. Each upgrade (i, r) has incremental bit cost $\Delta b_{i,r} > 0$ and incremental distortion reduction $\Delta e_{i,r} \geq 0$ (equivalently, marginal improvement in the Stage 1 objective). Feasibility requires* prefix closure*: if upgrade (i, r) is selected, then all (i, r') with $r' < r$ must also be selected. The upgrades along each chain are indexed so that marginal returns are nonincreasing:*

$$\frac{\Delta e_{i,1}}{\Delta b_{i,1}} \geq \frac{\Delta e_{i,2}}{\Delta b_{i,2}} \geq \cdots \geq \frac{\Delta e_{i,K_i}}{\Delta b_{i,K_i}}. \tag{19}$$

Theorem A.2 (Equal-cost exchange and optimality for a single chain) *Consider the problem of maximizing total distortion reduction $\sum_{i,r} x_{i,r}\, \Delta e_{i,r}$ subject to $\sum_{i,r} x_{i,r}\, \Delta b_{i,r} \leq B$, where $x_{i,r} \in \{0, 1\}$ respects Definition 1. Assume Eq. 19 holds for every matrix i. Write $\rho_{i,r} := \Delta e_{i,r} / \Delta b_{i,r}$.*

(Equal-cost exchange inequality.) *Assume $\Delta b_{i,r} \equiv \Delta > 0$ for all (i, r). Let $(k_1, \dots, k_n)$ encode a feasible prefix selection with $\sum_i k_i \leq K = \lfloor B/\Delta \rfloor$. If there exist chains $i \neq j$ with $k_i < K_i$, $k_j \geq 1$, and $\rho_{i,k_i+1} > \rho_{j,k_j}$, then replacing the terminal upgrade (j, k_j) by $(i, k_i + 1)$ increases $\sum_{p,r} x_{p,r} \Delta e_{p,r}$ while preserving feasibility and total bit cost.*

(Optimality only when $n = 1$.) *If $n = 1$, the greedy rule that repeatedly selects the next feasible upgrade with largest ratio $\rho_{1,r}$ until the budget is exhausted maximizes $\sum_r x_{1,r} \Delta e_{1,r}$ under the bit constraint (for arbitrary positive incremental costs $\Delta b_{1,r}$, since within a single chain comparing successive ratios orders upgrades consistently with Eq. 19). For $n > 1$, this greedy rule is* not *asserted to be globally optimal; see the remark below.*

Proof. *Equal-cost exchange.* Under uniform Δ, removing (j, k_j) frees one budget unit and adding $(i, k_i + 1)$ consumes one unit, leaving $\sum_{p,r} x_{p,r} \Delta b_{p,r}$ unchanged. The objective changes by $\Delta e_{i,k_i+1} - \Delta e_{j,k_j} = \Delta(\rho_{i,k_i+1} - \rho_{j,k_j}) > 0$.

Single-chain case ($n = 1$). Feasible solutions are prefixes $\{1, \dots, k\}$ for $k \in \{0, \dots, K_1\}$. Let $V(k) = \sum_{r \leq k} \Delta e_{1,r}$ and $B(k) = \sum_{r \leq k} \Delta b_{1,r}$. After Pareto filtering we may assume $\Delta e_{1,r} \geq 0$ on all retained upgrades, hence $V(k)$ is nondecreasing in k. Maximizing $V(k)$ subject to $B(k) \leq B$ therefore reduces to choosing the largest feasible k, namely $k^\star = \max\{k \in \{0, \dots, K_1\} : B(k) \leq B\}$. The sequential greedy construction that adds upgrades in increasing r until the next upgrade violates the budget outputs exactly this maximal feasible prefix. □

Remark (multi-chain coupling: heuristic greedy, no approximation ratio). For $n > 1$, even with uniform Δ, repeatedly choosing the frontier upgrade with largest ρ among all matrices need not yield a globally optimal integer solution: feasible sets have Cartesian product structure with per-chain prefixes, and the resulting discrete problems fall into precedence-constrained knapsack families that are NP-hard in general (Kellerer et al., 2004). We therefore do *not* interpret Stage 1's Eq. 7 greedy loop as an optimal or approximately optimal solver in the worst case. Theorem A.2's exchange inequality still captures the canonical *local* improvement under uniform incremental costs and Eq. 19: whenever its hypotheses apply, swapping terminal upgrades across chains strictly increases total distortion reduction while holding bit usage fixed. Deriving a constant-factor approximation (for example, via verified submodularity of the objective over feasible prefixes) would require additional assumptions tailored to the empirical $\Delta e_{i,r}$ and is left for future work.

Table 7: Multi-task adaptation results on Llama-3-8B.

Method	Trainable Params	GSM8K Acc. ↑	SAMSum ↑	BlendX ↑
AQLM + LoRA ($r = 36$)	94.37MB	25.70%	**0.3303**	0.4214
AQLM + Native Adapter	94.00MB	21.23%	0.3156	0.4398
AMPCQ + LoRA ($r = 36$)	94.37MB	**40.13%**	0.3294	0.4273
AMPCQ + Native Adapter (Ours)	94.00MB	30.78%	0.3298	**0.4421**

A.3 FORMAL BIT ACCOUNTING IN AMPCQ

This subsection formalizes per-matrix storage (in bits) used in Stage 1 and aligns with Section 3.2. For weight matrix $W_i \in \mathbb{R}^{d_{\text{out},i} \times d_{\text{in},i}}$, let its assigned tier be

$$\tau_i = (K_i, S_i, g_{\text{out},i}, g_{\text{in},i}). \tag{20}$$

The number of quantized blocks is

$$N_i(\tau_i) = \frac{d_{\text{out},i}}{g_{\text{out},i}} \cdot \frac{d_{\text{in},i}}{g_{\text{in},i}}. \tag{21}$$

Under this tier, total storage (in bits) is decomposed as

$$R_i(\tau_i) = R_i^{\text{index}}(\tau_i) + R_i^{\text{codebook}}(\tau_i) + R_i^{\text{scale}}(\tau_i), \tag{22}$$

with

$$R_i^{\text{index}}(\tau_i) = N_i(\tau_i)\, K_i\, \log_2 S_i, \tag{23}$$

$$R_i^{\text{codebook}}(\tau_i) = K_i\, S_i\, g_{\text{out},i}\, g_{\text{in},i} \cdot 16, \tag{24}$$

$$R_i^{\text{scale}}(\tau_i) = d_{\text{out},i} \cdot 16. \tag{25}$$

The scale term follows the AMPCQ implementation: one FP16 scale per output channel. Therefore,

$$\begin{aligned} R_i(\tau_i) = N_i(\tau_i)\, K_i\, \log_2 S_i \\ + K_i\, S_i\, g_{\text{out},i}\, g_{\text{in},i} \cdot 16 \\ + d_{\text{out},i} \cdot 16. \end{aligned} \tag{26}$$

The resulting model-average bit width (bits per scalar weight) is

$$\bar{b}(\{\tau_i\}) = \frac{\sum_{i=1}^{n} R_i(\tau_i)}{\sum_{i=1}^{n} d_{\text{out},i}\, d_{\text{in},i}}. \tag{27}$$

This is equivalent to Eq. 5 by choosing $\alpha_i = d_{\text{out},i} d_{\text{in},i}$ and $B = \bar{b}_{\text{target}} \sum_i \alpha_i$, giving

$$\sum_{i=1}^{n} \alpha_i R_i(\tau_i) \le B \iff \bar{b}(\{\tau_i\}) \le \bar{b}_{\text{target}}. \tag{28}$$

B REPRODUCIBILITY AND EXTENDED EXPERIMENTAL DETAILS

B.1 SOFTWARE AND COMPUTE ENVIRONMENT

All experiments run on NVIDIA A800 GPUs (80 GB). Unless otherwise noted, AMPCQ timing and training jobs use a single GPU. We use multiple GPUs only for tensor parallelism or device placement when model size requires it; quantization hyperparameters are unchanged across placements. We implement AMPCQ in Python 3.10 or newer with PyTorch (Compute Unified Device Architecture, CUDA 12.x), `transformers`, and `datasets` for model loading and data access. Zero-shot tasks use the Language Model Evaluation Harness (Gao et al., 2024); we pin the harness commit or release tag in the code release.

Table 8: **Stage 1 ablation** (Llama3-8B): ROI ranking *NMSE only* (ignore bit cost in the marginal score) vs. default bit-aware ROI (Eq. 7). Stage 2 recovery disabled; WikiText-2/C4 perplexity and mean five-task accuracy.

Method	Avg_bits	Wiki2 ↓	C4 ↓	Avg_acc ↑
NMSE only	2.27	24.52	37.93	57.12
Use bit cost	2.27	**15.39**	**28.61**	**58.64**
NMSE only	3.04	8.77	15.35	64.17
Use bit cost	3.04	**7.78**	**13.80**	**64.61**

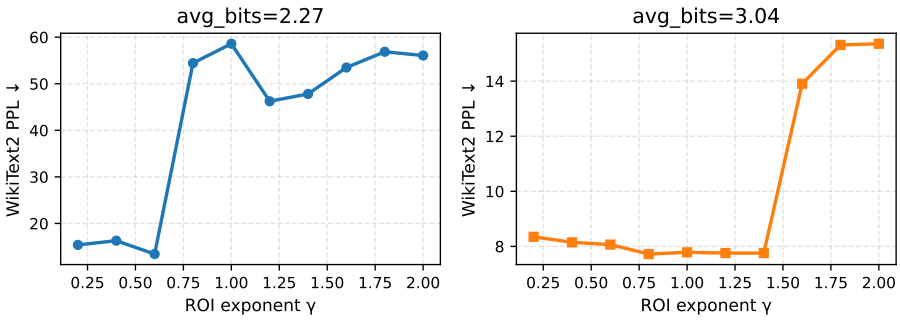

Figure 2: **Stage 1 ablation** (Llama3-8B): WikiText-2 perplexity vs. ROI exponent γ in Eq. 7 at average bit widths 2.27 and 3.04 bits.

B.2 Randomness and Calibration Splits

We fix pseudorandom seeds for (i) subsampling the 256 WikiText-2 training sequences used for activation and Hessian statistics, and (ii) any stochastic components in baseline optimizers when the upstream implementation exposes them. Calibration text is drawn contiguously from WikiText-2 `train`, then truncated and packed to each base model's default tokenizer context length unless GPU memory forces a shorter window (recorded in the released configuration).

B.3 AMPCQ Defaults and Training Schedule

Unless otherwise specified in the paper or released configs, we use block grouping g=8 (Table 1), Adam with learning rate 10^{-3} for codebook parameters, beam width 8 for dual-codebook discrete search, and Stage 1 ROI exponent γ=1 (Eq. 7). The same 256 WikiText-2 *training* sequences supply activation and Hessian statistics for Stage 1 profiling and Stage 2 layer-wise recovery; seeds and packing are fixed as in Section B.2.

Table 1 lists the discrete quantization tiers for Stage 1 profiling and Stage 2 recovery. When tiers are not fixed by an ablation, codebooks are optimized for the step budget shipped with our code; we do not use learning-rate warmup by default. Adapter or task-specific fine-tuning runs reuse the same calibration protocol unless the configuration states otherwise.

B.4 Additional Ablations for Section 5.2.3

This subsection collects additional controlled experiments for the Stage 1 design. Table 8 isolates bit-cost-aware ROI ranking, Figure 2 varies the ROI exponent γ, and Table 9 studies the number of profiling sequences. We further compare alternative allocation strategies under an identical VQ candidate space, examine whether Stage 2 recovery capacity changes the relative behavior of different tier spaces, and evaluate a gradient-weighted sensitivity metric.

Comparison with alternative allocation strategies. To isolate the contribution of the allocation strategy, we adapt the allocation principles of LLM-MQ (Li et al., 2023), SliM-LLM (Huang et al., 2025), and Q-Palette (Lee & Song, 2025) to the same VQ candidate pool. All methods use the

Table 9: **Stage 1 ablation** (Llama3-8B): WikiText-2/C4 perplexity and mean zero-shot accuracy vs. number of WikiText-2 `train` sequences used for profiling statistics at 3.04-bit average width.

Profiling samples	Wiki2 ↓	C4 ↓	Avg_acc ↑
64	8.25	13.90	64.05
128	8.26	13.92	63.95
256	**7.78**	**13.80**	**64.61**

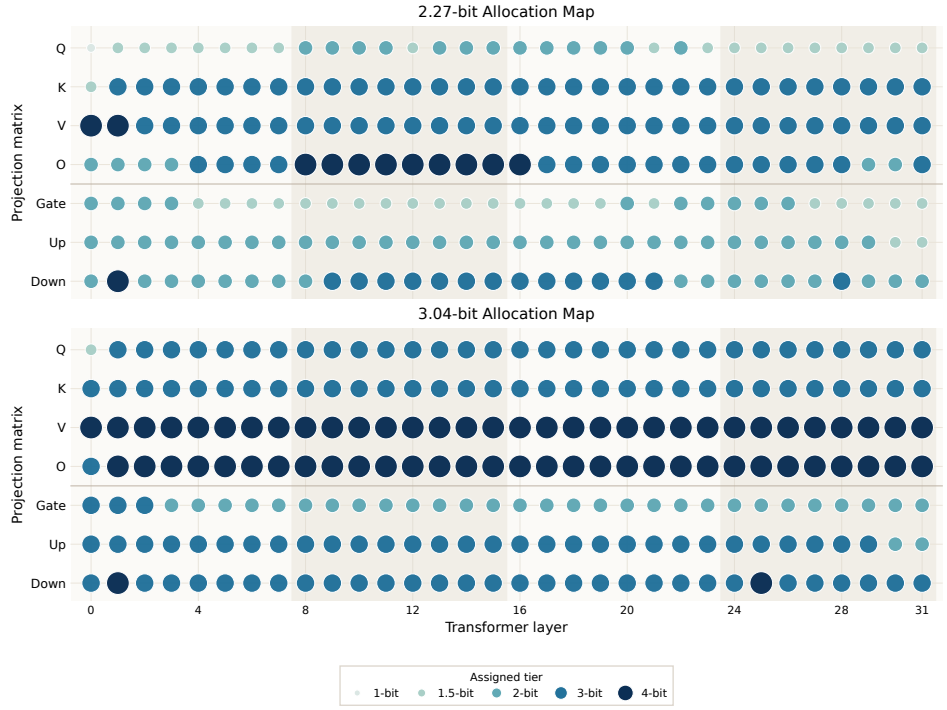

Figure 3: Stage 1 matrix-wise tier allocation maps on Llama3-8B under 2.27-bit and 3.04-bit average budgets. Each marker corresponds to one attention or MLP projection matrix in one transformer layer, with size and color indicating the selected AMPCQ tier.

same Llama3-8B model, candidate codebook tiers, calibration data, storage-cost model, and ≈2.27-bit average budget, with Stage 2 recovery disabled. This controlled setting isolates the allocator from the underlying quantizer. As shown in Table 10, AMPCQ achieves the highest mean zero-shot accuracy among the compared allocation strategies.

Robustness to Stage 2 recovery capacity. Stage 1 evaluates candidate tiers before the full Stage 2 recovery procedure, raising the question of whether different tier spaces may benefit unevenly from subsequent optimization. We therefore select 12 MLP matrices (gate, up, and down projections from layers 0, 10, 21, and 31) of Llama3-8B and compare a restricted $\{2, 4\}$-bit tier space against the full $\{1, 1.5, 2, 3, 4\}$-bit tier space under nearly identical average-bit budgets. Both configurations use the same quantizer, calibration data, and Stage 2 recovery schedule. As shown in Table 11, the two tier spaces exhibit different distributions of index and codebook parameters, but their relative performance changes only marginally after Stage 2 recovery. This suggests that differences in recovery capacity do not materially alter the Stage 1 comparison.

Gradient-based sensitivity. We additionally compare the default activation-normalized reconstruction metric with a gradient-weighted sensitivity measure inspired by SqueezeLLM (Kim et al., 2024). Both variants use the same Llama3-8B model and ≈2.27-bit average budget. As shown in Table 12, gradient weighting reduces mean zero-shot accuracy from 58.64 to 56.58. These results empirically support activation-normalized reconstruction as the profiling metric in our setting.

Table 10: Comparison of allocation strategies under the same codebook quantizer and ≈2.27-bit average budget on Llama3-8B. Stage 2 recovery is disabled to isolate allocation quality.

Method	PIQA	ARC-E	ARC-C	HellaSwag	WinoGrande	Avg. ↑
LLM-MQ-style (Li et al., 2023)	54.24	28.11	21.16	27.27	51.30	36.42
SliM-inspired (Huang et al., 2025)	69.48	61.15	27.47	37.99	62.04	51.63
Q-Palette VQ (Lee & Song, 2025)	73.01	65.87	35.32	45.65	**67.17**	57.40
AMPCQ	**73.72**	**69.32**	**37.20**	**46.88**	66.06	**58.64**

Table 11: Stage 2 recovery under different tier spaces on 12 Llama3-8B MLP matrices. Parameter counts denote the updatable quantities during recovery.

Tier Space	Avg. bits	Codebook Params	Scale Params	Stage 1 Avg.	Stage 2 Avg.
$\{2, 4\}$	3.2143	9.44M	0.13M	68.75	**69.04**
$\{1, 1.5, 2, 3, 4\}$	3.2217	6.09M	0.13M	**68.79**	68.97

B.5 BASELINE ALIGNMENT

AQLM / uniform codebooks. We follow each method's public recipe for codebook size, grouping g=8, and optimization budget, then vary only the target average bit width to match each table row.

GPTQ / AWQ / RTN / QuIP# / PocketLLM. Each baseline is built from its reference implementation or the best-maintained public fork. Whenever supported, we align grouping, symmetric versus asymmetric bit assignments, and calibration sample counts with our WikiText-2 protocol; if a baseline defaults to another corpus, we document the mismatch and the reason in the released configuration.

B.6 PERPLEXITY AND ZERO-SHOT EVALUATION

Perplexity. WikiText-2 and C4 use standard causal language-model perplexity on the splits distributed with each dataset (validation or test, per dataset convention).

Zero-shot accuracy. We run `winogrande`, `piqa`, `hellaswag`, `arc_easy`, and `arc_challenge` in the harness under 0-shot prompts and report the arithmetic mean as **Avg_acc**. Batching and device placement are held consistent across methods; we ship exact command lines with the code release.

B.7 MULTI-TASK BENCHMARK (SECTION 5.2.2)

Protocols match Section 5.2.2. AQLM and PocketLLM under matched average bit width, the Stage 1–2 AMPCQ backbone, and an optional Stage 3 codebook-delta row labeled *(Tuning)*. BlendX (intent), SAMSum (summarization), and GSM8K (math) each use that benchmark's primary automatic metric (e.g., accuracy, F1 score, or Recall-Oriented Understudy for Gisting Evaluation (ROUGE)-based scores) as defined by its reference evaluator. We report the same headline fields as those scripts and do not rescale scores post hoc.

B.8 VISUALIZATION OF MATRIX-WISE TIER ALLOCATION

Figure 3 visualizes the Stage 1 matrix-wise tier assignments on Llama3-8B under two representative average-bit budgets. The learned allocation is non-uniform across both transformer depth and projection type. Under the tighter 2.27-bit budget, AMPCQ selectively preserves higher tiers for a subset of projections, especially several K/V/O and Down matrices, while assigning lower tiers to many Q, Gate, and Up matrices. When the budget increases to 3.04 bits, high-capacity tiers are assigned more broadly, but the allocation remains structured rather than uniform. This provides a qualitative view of the budget-dependent behavior behind the mixed-tier gains reported in Table 4.

Table 12: Activation-based versus gradient-weighted sensitivity on Llama3-8B at ≈2.27-bit average width.

Metric	PIQA	ARC-E	ARC-C	HellaSwag	WinoGrande	Avg. ↑
Activation NMSE	**73.72**	**69.32**	**37.20**	46.88	**66.06**	**58.64**
Gradient-weighted (Kim et al., 2024)	72.63	65.24	33.11	**47.50**	64.40	56.58

Table 13: **Inference micro-benchmark**: single-token MLP forward time for one Llama3-8B decoder layer on RTX 3080 Ti.

Module	FP16 Llama (μs) ↓	AMPCQ (μs) ↓	Ratio ↑
mlp	781.09	**743.02**	1.05
mlp.gate_proj	201.84	184.08	1.10
mlp.up_proj	193.84	168.37	1.15
mlp.down_proj	193.86	163.47	1.19

B.9 EDGE-DEVICE INFERENCE LATENCY VERSUS FP16.

We time single-token MLP forward passes on one Llama3-8B decoder layer on an RTX 3080 Ti (12 GB), batch size 1, sequence length 1, FP16 decode. We call `mlp` and submodules `gate_proj`, `up_proj`, and `down_proj`. Table 13 reports measured latency (μs) and ratio (FP16 time divided by AMPCQ time); ratios >1 correspond to shorter AMPCQ wall times in this benchmark.

C ABLATION STUDY

We perform additional validation of the Stage 1 design using Qwen3-14B, following the same protocol established for Llama3-8B. Profiling, tier allocation, ROI ranking, and evaluation follow the Llama3-8B protocol; only the pretrained checkpoint changes.

We report WikiText-2 and C4 perplexity and mean zero-shot accuracy over ARC-Easy, ARC-Challenge, HellaSwag, PIQA, and WinoGrande. The dense baseline models are evaluated using 16-bit floating point (FP16); all quantized models share the same calibration draw.

Figure 4 plots WikiText-2 perplexity versus γ at 3.02 and 3.80 bits. Table 14 shows that mixed tiers yield lower perplexity and higher mean accuracy compared to a uniform tier at 3.02 bits. Table 15 indicates that activation NMSE achieves the lowest perplexity and highest mean accuracy among the four distortion metrics evaluated. Table 16 compares ROI ranking based only on NMSE against the default bit-aware ROI. At 3.02 bits, WikiText-2 perplexity is 9.67 for both rows; C4 perplexity is lower for NMSE-only (17.23 vs. 17.52), while mean accuracy is higher for bit-aware ROI (67.60 vs. 67.25). At 3.80 bits, NMSE-only attains lower perplexity on both corpora (8.66/15.77 vs. 8.72/15.80), while bit-aware ROI attains higher mean accuracy (70.21 vs. 69.75).

Table 14: **Stage 1 ablation** (Qwen3-14B): uniform tier vs. mixed matrix-wise tiers at the same **Avg_bits**; Stage 2 identical across rows.

Method	Avg_bits	Wiki2 ↓	C4 ↓	Avg_acc ↑
Uniform	3.02	11.18	20.27	65.34
Mixed	3.02	**9.67**	**17.52**	**67.60**

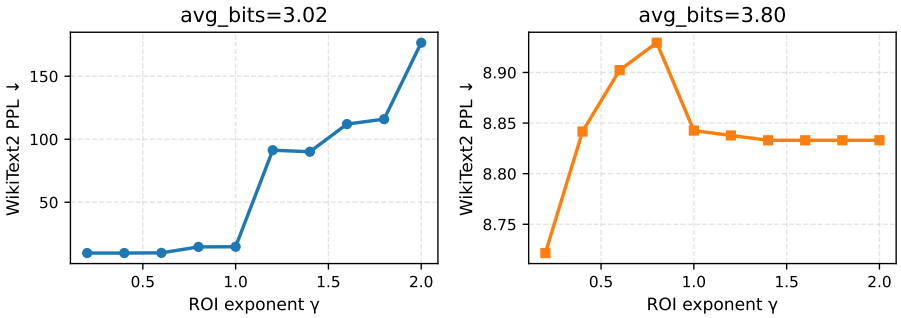

Figure 4: **Stage 1 ablation** (Qwen3-14B): WikiText-2 perplexity vs. γ (Eq. 7) at average bit widths 3.02 and 3.80 bits (protocol aligned with Llama3-8B ablations).

Table 15: **Stage 1 ablation** (Qwen3-14B): distortion metric for scoring tier candidates (Section 3.2); fixed Stage 1 budget and Stage 2 pipeline.

Allocation metric	Wiki2 ↓	C4 ↓	Avg_acc ↑
Plain MSE	11.04	20.22	65.38
Plain NMSE	10.91	19.75	65.09
Activation MSE	13.23	23.64	57.07
Activation NMSE	**9.67**	**17.52**	**67.60**

Table 16: **Stage 1 ablation** (Qwen3-14B): *NMSE only* vs. bit-aware ROI (Eq. 7).

Method	Avg_bits	Wiki2 ↓	C4 ↓	Avg_acc ↑
NMSE only	3.02	9.67	**17.23**	67.25
Use bit cost	3.02	**9.67**	17.52	**67.60**
NMSE only	3.80	**8.66**	**15.77**	69.75
Use bit cost	3.80	8.72	15.80	**70.21**