AAAC: ACTIVATION-AWARE ADAPTIVE CODEBOOKS FOR 4-BIT LLM WEIGHT QUANTIZATION

Anonymous authors
Paper under double-blind review

ABSTRACT

Post-training weight-only quantization (PTQ) to 4 bits is widely used to reduce the memory and compute costs of large language model inference. Existing PTQ methods, such as AWQ and GPTQ, improve how weights are mapped onto a fixed 4-bit grid through scaling, clipping, or error compensation. To further improve accuracy, methods such as OmniQuant and QuIP# uses gradient-assisted algorithms at the cost of hours of quantization time. In this work, we propose AAAC (Activation-Aware Adaptive Codebooks), a lightweight method for 4-bit LLM weight quantization. AAAC replaces the fixed scalar codebook used in standard quantization with two small learned scalar codebooks (64 bytes) per layer. Each group of weights selects the codebook that minimizes activation-weighted reconstruction error, encoding the choice in the unused sign bit of the group's positive scale and adding zero storage overhead. AAAC completes in 3–30 minutes on a single GPU, and adds no memory beyond the model itself. We evaluate against AWQ, GPTQ, IF4, GPTVQ, OmniQuant, SqueezeLLM, and QuIP# across model families. AAAC outperforms baselines at orders-of-magnitude less quantization time.

1 INTRODUCTION

Large language models (LLMs) are widely deployed under memory and compute constraints that make weight quantization essential (Touvron et al., 2023; Qwen et al., 2025). Weight-only 4-bit quantization substantially reduces model storage and memory bandwidth while retaining acceptable quality for many models (Dettmers et al., 2023).

Most post-training quantization (PTQ) methods improve how weights are mapped onto a fixed 4-bit grid. In group-wise quantization, each group of weights shares a scale s, and each normalized weight $\tilde{w} = w/s$ is rounded to a discrete code under a format-defined table $\mathcal{T}$. The same table is then used during dequantization:

$$c = Q(w;\, s), \qquad \hat{w} = \mathcal{T}[c] \cdot s. \tag{1}$$

For INT4, $\mathcal{T}$ is an integer grid; for NVFP4, $\mathcal{T}$ is the FP4 grid. Once the format is chosen, this scalar grid is usually treated as fixed. Methods such as AWQ and GPTQ therefore improve quantization around this fixed grid, using channel scaling (Lin et al., 2024), clipping, or error compensation (Frantar et al., 2023), while leaving the scalar reconstruction grid itself unchanged. This suggests a complementary PTQ direction: instead of only asking how to map weights onto a fixed FP4 or INT4 table, learn adaptive scalar codebooks that better match the weight distribution and the activation statistics of the layer.

We propose **Activation-Aware Adaptive Codebooks (AAAC)**, a lightweight method for 4-bit LLM weight quantization. AAAC replaces the single fixed table $\mathcal{T}$ with two learned scalar codebooks per layer, $\mathcal{T}_0$ and $\mathcal{T}_1$. Each group of weights selects the codebook that minimizes activation-weighted reconstruction error, and weights in that group are stored as ordinary 4-bit indices into the selected codebook. AWQ and GPTQ improve how weights are mapped onto a fixed 4-bit table; AAAC improves the table used for that mapping.

AAAC distinguishes itself from prior PTQ methods on three axes. **Negligible overhead**: AAAC adds only two BF16 codebooks (64 bytes) per layer. The per-group selection bit is stored in the unused sign of the always-positive scale, adding zero storage. **No gradients**: codebooks are learned via

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.

activation-weighted k-means on forward-pass activations alone—no backpropagation, no Hessian or Fisher computation. **No memory blowup**: AAAC processes each layer independently using only its activations and codebook statistics, so peak memory stays close to the model size itself.

2 Related Work

Post-training quantization methods for LLMs can be divided into gradient-free approaches, which use only forward-pass calibration data, and gradient-assisted approaches, which incur larger quantization times and/or peak memory overhead.

Gradient-free PTQ. GPTQ (Frantar et al., 2023) quantizes weights column-by-column using inverse-Hessian error compensation. AWQ (Lin et al., 2024) protects salient weights using channel scaling determined. IF4 (Cook et al., 2026) selects per-group between FP4 and INT4 codebooks based on MSE. All complete in minutes with peak memory close to the original model size.

Gradient-assisted PTQ. OmniQuant (Shao et al., 2024) learns per-channel clipping thresholds and equivalent transforms via block-wise backpropagation through the rounding operation, requiring 1–16 hours for 7B–70B models.

SqueezeLLM (Kim et al., 2024) learns per-channel non-uniform lookup tables via Fisher-weighted k-means, where the Fisher diagonal is computed by backpropagating through the full model. While the algorithm runs in 11–80 minutes, the Fisher step drives peak memory to 2–4$\times$ model size.

QuIP# (Tseng et al., 2024) applies a Randomized Hadamard Transform for incoherence processing, then rounds to a fixed E8 lattice codebook via BlockLDLQ. Without fine-tuning this is gradient-free and completes in under 10 GPU-hours for 70B. An optional fine-tuning stage adds backpropagation through the full model, raising the cost to roughly 100 GPU-hours.

GPTVQ (van Baalen et al., 2025) extends GPTQ to vector quantization, learning per-group codebooks via Hessian-weighted EM followed by gradient descent on the layer-wise reconstruction objective to refine codebook entries. Quantization takes 0.5–11 hours for 7B–70B models.

Our experiments show that gradient-assisted methods consistently outperform gradient-free methods, suggesting that gradients—with their associated quantization time and memory overhead—are necessary to close the gap to full-precision quality. AAAC challenges this conclusion: it is a lightweight, gradient-free method that learns only two codebooks per layer (64 bytes overhead) via activation-weighted k-means. Despite this simplicity, AAAC outperforms all gradient-free methods and matches or exceeds gradient-assisted methods, with 3–30 minutes of quantization time and peak memory no larger than the original model size.

3 Method

3.1 Background: Group-wise Quantization

Consider a linear layer computing $\mathbf{y} = \mathbf{x}\mathbf{W}^\top$ with $\mathbf{W} \in \mathbb{R}^{N\times K}$. In group-wise quantization, each row of $\mathbf{W}$ is partitioned into groups of g contiguous elements. Each group shares a scale $s > 0$, and each weight is represented by a discrete code.

Quantization. A scale s is computed using weight statistics. Each weight w is normalized by s, rounded to the nearest entry in the format-defined table $\mathcal{T}$, and stored as a b-bit code:

$$\tilde{w} = w/s, \qquad c = \arg\min_i |\tilde{w} - \mathcal{T}[i]|. \tag{2}$$

Dequantization. The stored code is looked up in the same table and multiplied by the scale:

$$\hat{w} = \mathcal{T}[c] \cdot s. \tag{3}$$

Thus the table $\mathcal{T}$ plays two roles: it defines the nearest-neighbor code assignment during quantization, and it defines the reconstruction value used during dequantization.

It is useful to distinguish the stored code from the reconstructed value. For any scalar table $\mathcal{A}$, define the nearest-entry reconstruction operator

$$\mathrm{recon}_{\mathcal{A}}(\tilde{w}) = \mathcal{A}\Big[\arg\min_i |\tilde{w} - \mathcal{A}[i]|\Big]. \tag{4}$$

This operator first finds the nearest entry of $\mathcal{A}$ to the normalized weight $\tilde{w}$, then returns the corresponding table value. Under the base format, the reconstructed normalized weight is $\mathrm{recon}_{\mathcal{T}}(\tilde{w})$, and the dequantized weight after quantization is $\mathrm{recon}_{\mathcal{T}}(\tilde{w}) \cdot s$.

The table $\mathcal{T}$ is fixed by the format. For NVFP4, $\mathcal{T}$ contains the signed E2M1 values used by the FP4 format, such as $\{-6, -4, \ldots, 0, \ldots, 3, 4, 6\}$. For INT4, $\mathcal{T}$ denotes the signed or zero-point-adjusted integer grid used by the implementation, expressed in normalized units. Thus both NVFP4 and INT4 can be written as group-wise scalar quantizers with a finite code alphabet and a per-group scale.

Key observation. Existing PTQ methods such as AWQ and GPTQ primarily improve quantization around the fixed table $\mathcal{T}$. They change the weight distribution, scales, or compensation procedure so that the fixed table produces better reconstructed values. AAAC targets a different degree of freedom: it learns the scalar table itself.

3.2 AAAC: Activation-Aware Adaptive Codebooks

AAAC replaces the single fixed table $\mathcal{T}$ with two learned scalar codebooks $\mathcal{T}_0$ and $\mathcal{T}_1$. Each learned table has the same number of entries as the base 4-bit format, but its entries are optimized from calibration data rather than fixed by the format. A group of S consecutive weights selects one of the two tables. The selected table defines the nearest-entry reconstruction used for that group.

Granularity parameters. AAAC has two granularity parameters. The *scale group size* g is fixed by the quantization format and determines how many weights share one scale: g=16 for NVFP4 and g=128 for INT4. The *selection group size* S determines how many weights share one codebook choice. Thus, every S weights store one selection bit σ, and every g weights store one scale.

When S=g, the selection group coincides with the scale group: each scale needs exactly one codebook-selection bit. Since the scale is always positive, its sign bit is unused, and we store σ there with no additional selection storage. When S<g, the selection costs $1/S$ bits per weight. Smaller S gives finer codebook adaptation. We use S=g=16 for NVFP4 and evaluate $S \in \{16, 128\}$ for INT4.

Modified quantization and dequantization. For a weight w in selection group j, let $\tilde{w} = w/s$. AAAC selects table $\mathcal{T}_{\sigma_j}$ and reconstructs the normalized weight using the nearest-entry reconstruction:

$$\hat{w} = \mathrm{recon}_{\mathcal{T}_{\sigma_j}}(\tilde{w}) \cdot s. \tag{5}$$

Equivalently, the stored 4-bit code is the nearest-entry index under the selected table:

$$c = \arg\min_i |\tilde{w} - \mathcal{T}_{\sigma_j}[i]|. \tag{6}$$

Thus AAAC preserves the scalar 4-bit code width, group-wise scale structure, and packed representation, but it does not require the code assignment to match round-to-nearest under the original table $\mathcal{T}$. Its improvement comes from learning better scalar reconstruction grids and choosing, per group, which grid to use.

Activation importance. For a linear layer $\mathbf{y} = \mathbf{x}\mathbf{W}^\top$, a perturbation $\delta\mathbf{W}$ produces output error $\delta\mathbf{y} = \mathbf{x}\delta\mathbf{W}^\top$. The expected squared output error contributed by column k of $\delta\mathbf{W}$ is proportional to $\mathbb{E}[x_k^2]\|\delta\mathbf{W}_{:,k}\|^2$. Summing over a calibration set $\mathbf{X} \in \mathbb{R}^{T\times K}$ gives the per-column importance

$$I_k = \sum_{t=1}^{T} X_{t,k}^2. \tag{7}$$

Minimizing $\sum_k I_k \|\delta\mathbf{W}_{:,k}\|^2$ is therefore a diagonal proxy for minimizing calibration output error. Columns with larger activation energy receive larger weight in the reconstruction objective.

Table selection. Consider a selection group j containing normalized weights $\{\tilde{w}_{j,1}, \ldots, \tilde{w}_{j,S}\}$ and activation importances $\{I_{j,1}, \ldots, I_{j,S}\}$. AAAC assigns the group to the table that minimizes activation-weighted reconstruction error:

$$\sigma_j = \arg\min_{r\in\{0,1\}} \sum_{k=1}^{S} I_{j,k} \left(\tilde{w}_{j,k} - \mathrm{recon}_{\mathcal{T}_r}(\tilde{w}_{j,k})\right)^2. \tag{8}$$

The table choice and code assignment are coupled: each candidate table defines both the reconstruction values and the nearest-entry regions used to encode the weights in the group.

Storage. Each learned table contains $|\mathcal{T}|$ values stored in BF16, for a total of $2|\mathcal{T}| \cdot 2$ bytes per layer: 60 bytes for a 15-entry NVFP4 table and 64 bytes for a 16-entry INT4 table. For a 7B model with $\sim$224 linear layers, this is about 14 KB, negligible compared with the model weights. The selection bits $\{\sigma_j\}$ require no additional storage when $S{=}g$ if they are stored in the unused sign bit of each positive scale, and require $1/S$ bits per weight when $S{<}g$.

3.3 Learning Algorithm

We now describe how the tables $\mathcal{T}_0$ and $\mathcal{T}_1$ are obtained from calibration data. The two tables are learned per layer using a small calibration set. Learning alternates between assigning selection groups to tables and refining the table entries. Algorithm 1 provides pseudocode.

Initialization. Let $M = |\mathcal{T}|$ be the number of entries in each table, and let $\mathrm{Quantile}_q(\{\tilde{w}\})$ denote the q-quantile of the normalized weight distribution. We initialize $\mathcal{T}_0$ using M evenly spaced quantiles spanning the full range:

$$\mathcal{T}_0[i] = \mathrm{Quantile}_{i/(M-1)}(\{\tilde{w}\}), \qquad i = 0, \ldots, M-1. \tag{9}$$

Initialize $\mathcal{T}_1$ using M evenly spaced quantiles starting at a half-step offset and ending at the maximum:

$$\mathcal{T}_1[i] = \mathrm{Quantile}_{\delta+(1-\delta)i/(M-1)}(\{\tilde{w}\}), \qquad \delta = \frac{1}{2(M-1)}. \tag{10}$$

This shifts the second grid forward by half a quantile spacing, giving the two tables distinct but nearby starting points. Results are insensitive to the exact offset.

Alternating optimization. Learning alternates between two steps for n_{outer} iterations. In the *assignment step*, each selection group is assigned to the table that minimizes its activation-weighted reconstruction error, as in Eq. 8. In the *table update step*, each table is refined by n_{inner} iterations of activation-weighted scalar k-means. For each table, normalized weights assigned to that table are assigned to their nearest table entry, and each entry is updated to the activation-weighted centroid of its Voronoi cell:

$$\mathcal{T}_r[i] \leftarrow \frac{\sum_{(\tilde{w},I)\in\mathcal{V}_{r,i}} I\tilde{w}}{\sum_{(\tilde{w},I)\in\mathcal{V}_{r,i}} I}. \tag{11}$$

Here, $\mathcal{V}_{r,i}$ is the set of normalized weights assigned to entry i of table $\mathcal{T}_r$. If a Voronoi cell is empty, we keep the entry at its previous value. After the final inner update, the entries of each table are sorted in ascending order.

The outer loop matters more than the inner loop: reassigning groups to tables has a larger effect on quality than refining table entries within a fixed assignment. We find empirically that a single outer iteration already achieves 82% of the total improvement, so the overall cost of the procedure is dominated by the assignment step rather than the inner k-means refinement.

Computational cost. Each outer iteration sweeps over the layer's weights twice: once to assign every selection group to its better-fitting table, and once to update each table by importance-weighted k-means over the weights assigned to it. Both passes are forward-only and reduce to a small number of importance-weighted reductions; there are no gradients, no Hessian-vector products, no end-to-end backpropagation, and the codebook size $M = |\mathcal{T}| \in \{15, 16\}$ is small enough that nearest-entry lookups are essentially free. The algorithm processes layers independently and stores only forward activations and codebook statistics, so peak GPU memory is dominated by holding the model itself plus a per-layer activation buffer. This enables in-memory calibration of a 27B model in roughly 10 minutes under the configuration ($n_{\mathrm{outer}}{=}3$, $n_{\mathrm{inner}}{=}10$).

Algorithm 1 AAAC: Learning activation-aware adaptive codebooks for one layer

Require: Weight matrix $\mathbf{W} \in \mathbb{R}^{N \times K}$, calibration activations $\mathbf{X} \in \mathbb{R}^{T \times K}$
Require: Group sizes g (scale), S (selection); iterations n_{outer}, n_{inner}
Compute scales s per group of g weights
Normalize weights: $\tilde{\mathbf{W}} \leftarrow \mathbf{W}/s$
Compute activation importance: $I_k \leftarrow \sum_t X_{t,k}^2$ for each column k
Initialize $\mathcal{T}_0$ from full-range quantiles of $\tilde{\mathbf{W}}$ {Eq. 9}
Initialize $\mathcal{T}_1$ from half-shifted quantiles of $\tilde{\mathbf{W}}$ {Eq. 10}
for $t = 1$ to n_{outer} **do**
  // Assignment step
  for each selection group j **do**
    $\sigma_j \leftarrow \arg\min_{r \in \{0,1\}} \sum_{k=1}^{S} I_{j,k} \left(\tilde{w}_{j,k} - \text{recon}_{\mathcal{T}_r}(\tilde{w}_{j,k})\right)^2$
  end for
  // Table update step
  for $r \in \{0, 1\}$ **do**
    Let $\mathcal{W}_r$ be the multiset of pairs $(\tilde{w}_{j,k}, I_{j,k})$ with $\sigma_j = r$
    for $\ell = 1$ to n_{inner} **do**
      for each entry i in $\mathcal{T}_r$ **do**
        Let $\mathcal{V}_{r,i} \subseteq \mathcal{W}_r$ be the Voronoi cell of entry i
        $\mathcal{T}_r[i] \leftarrow \dfrac{\sum_{(\tilde{w},I) \in \mathcal{V}_{r,i}} I\tilde{w}}{\sum_{(\tilde{w},I) \in \mathcal{V}_{r,i}} I}$
        If the denominator is zero, keep the previous value of $\mathcal{T}_r[i]$
      end for
    end for
    Sort $\mathcal{T}_r$ in ascending order
  end for
end for
Round $\mathcal{T}_0, \mathcal{T}_1$ to BF16 precision
Recompute $\{\sigma_j\}$ using Eq. 8
Pack 4-bit codes as nearest-entry indices into $\mathcal{T}_{\sigma_j}$
Store $\{\sigma_j\}$ in sign bits of s if $S{=}g$, or separately if $S{<}g$
return $\mathcal{T}_0, \mathcal{T}_1$, packed 4-bit codes, scales, and $\{\sigma_j\}$

4 EXPERIMENTS

4.1 SETUP

Models. We evaluate on models spanning Llama (3.2-1B, 3.2-3B, 3.1-8B) (Grattafiori et al., 2024), Qwen2.5 (0.5B, 7B, 14B, 32B) (Qwen et al., 2025), and Qwen3.5 (2B, 4B, 9B, 27B). To compare with numbers reported by prior methods, we additionally evaluate on LLaMA-1 (7B, 13B) and LLaMA-2 (7B, 13B) (Touvron et al., 2023).

Evaluation. We report perplexity on WikiText-2 (Merity et al., 2016) and C4 (Raffel et al., 2023). For WikiText-2, we follow the GPTQ protocol: the full test set is tokenized without filtering and split into non-overlapping 2048-token segments, with per-token cross-entropy computed via manual logit shifting (Frantar et al., 2023). For C4, we use the validation split with up to 256 segments. We additionally evaluate on downstream tasks using lm-evaluation-harness (Gao et al., 2024).

Gap recovery. To summarize the overall effectiveness of each method, we report *gap recovery*: the percentage of the quantization-induced degradation that a method eliminates relative to the round-to-nearest baseline. For perplexity (lower is better), gap recovery is defined as $(\text{PPL}_{\text{RTN}} - \text{PPL}_{\text{method}})/(\text{PPL}_{\text{RTN}} - \text{PPL}_{\text{full}}) \times 100\%$, where PPL_{RTN} is the round-to-nearest perplexity and PPL_{full} is the full-precision baseline. For accuracy (higher is better), the analogous formula is $(\text{Acc}_{\text{method}} - \text{Acc}_{\text{RTN}})/(\text{Acc}_{\text{full}} - \text{Acc}_{\text{RTN}}) \times 100\%$. A recovery of 0% means no improvement over RTN; 100% means full-precision quality is restored.

Table 1: g=16 W4A16 perplexity on WikiText-2 (↓). AAAC uses S=g=16 (zero overhead). Best quantized result in **bold**.

Model	BF16	RTN	IF4	AWQ	AAAC
Llama-3.2-1B	9.71	10.68	10.61	10.46	**10.21**
Llama-3.2-3B	7.77	8.19	8.13	8.11	**7.99**
Llama-3.1-8B	6.19	6.56	6.51	6.51	**6.39**
Qwen3.5-2B	12.06	13.21	12.70	12.56	**12.37**
Qwen3.5-4B	9.42	10.25	10.21	10.16	**9.92**
Qwen3.5-9B	8.51	9.11	9.38	8.91	**8.56**
Qwen3.5-27B	6.80	7.02	6.92	7.03	**6.88**
Average	*8.64*	*9.29*	*9.21*	*9.11*	***8.90***
Recovery	—	—	*12.3%*	*28.1%*	***59.2%***

AAAC configurations. A smaller selection granularity S allows AAAC to better adapt to local weight structure within each layer. The NVFP4 memory layout groups weights into blocks of 16 with shared FP8 scales, and this g=16 setting is ideal for AAAC: 16 weights is small enough for fine-grained codebook selection, and the selection bit can be stored in the scale sign at zero overhead. IF4 uses the same g=16 layout, and AWQ can be straightforwardly adapted to it. We report g=16 results alongside both methods. To compare with prior methods that use the standard INT4 layout with g=128, we also report S=g=128 (zero overhead). However, S=128 is too coarse for effective codebook selection, so we additionally report S=16, adding a negligible $1/16$=0.0625 bpw overhead. We use n_{outer}=3, n_{inner}=10 with 4 calibration sequences from C4. We used a H100 GPU.

Baselines and reproducibility. We compare against round-to-nearest (RTN), IF4 (Cook et al., 2026), AWQ (Lin et al., 2024), GPTQ (Frantar et al., 2023), GPTVQ (van Baalen et al., 2025), OmniQuant (Shao et al., 2024), SqueezeLLM (Kim et al., 2024), and QuIP# (Tseng et al., 2024). GPTQ numbers are taken from Lin et al. (2024). We implement IF4 following Cook et al. (2026): for each group, both FP4 and scaled INT4 quantizations are evaluated, and the option with lower MSE is selected. For AWQ at g=128, we use the official AutoAWQ package; for g=16, we implement their per-channel scaling algorithm ourselves as AutoAWQ does not support FP4-based quantization. GPTVQ, OmniQuant, SqueezeLLM, and QuIP# numbers are taken from their respective papers; we verify that our RTN and full-precision baselines match those reported, ensuring consistent evaluation protocols.

4.2 Comparison with Gradient-Free Methods

4.2.1 Group Size 16 (NVFP4 Memory Layout)

The NVFP4 memory layout stores weights in groups of 16 with shared FP8 scales, and several recent methods build on this layout. IF4 (Cook et al., 2026) selects per-group between the native FP4 and a scaled INT4 codebook. AWQ (Lin et al., 2024), originally designed for INT4, can be straightforwardly adapted to this layout. AAAC uses S=g=16, storing the selection bit in the scale sign at zero overhead.

Table 1 presents WikiText-2 perplexity across 7 Llama and Qwen3.5 models. Across all models, AAAC achieves the lowest perplexity among all quantized methods. IF4 shows inconsistent behavior: on Qwen3.5-9B, it produces a perplexity of 9.38, worse than round-to-nearest at 9.11. AAAC remains robust across all architectures and model sizes.

4.2.2 Group Size 128 (INT4 Layout)

To compare with prior methods that use the standard INT4 layout with g=128, we evaluate AAAC at S=g=128 (zero overhead) and S=16 (+0.0625 bpw).

Comparison with AWQ. Table 2 compares AAAC against AWQ on Llama-3 and Qwen2.5 models. AWQ results are produced using the official AutoAWQ package.

Table 2: g=128 W4A16 WikiText-2 perplexity (↓) vs. AWQ. AWQ uses the official AutoAWQ package. **Bold**: best result per model.

				AAAC	
Model	FP16	RTN	AWQ	S=128	S=16
Llama-3.2-3B	7.81	8.49	8.26	8.16	**8.10**
Llama-3.1-8B	6.24	6.82	6.65	6.68	**6.56**
Qwen2.5-0.5B	13.07	15.54	15.01	14.47	**14.27**
Qwen2.5-7B	6.85	7.23	7.09	7.13	**7.07**
Qwen2.5-14B	5.29	5.78	5.69	5.65	**5.60**
Average	*7.85*	*8.77*	*8.54*	*8.42*	***8.32***
Recovery	—	—	*25.0%*	*38.0%*	***48.9%***

Table 3: g=128 W4A16 WikiText-2 perplexity (↓) vs. GPTQ. GPTQ numbers from Lin et al. (2024). **Bold**: best result per model.

				AAAC	
Model	FP16	RTN	GPTQ	S=128	S=16
LLaMA-1-7B	5.68	5.96	5.85	5.82	**5.76**
LLaMA-1-13B	5.09	5.25	5.23	5.21	**5.17**
LLaMA-2-7B	5.47	5.72	5.69	5.62	**5.58**
LLaMA-2-13B	4.88	4.98	4.98	4.98	**4.94**
Average	*5.28*	*5.48*	*5.44*	*5.41*	***5.36***
Recovery	—	—	*20.0%*	*35.0%*	***60.0%***

With S=g=128 (zero overhead), AAAC already outperforms AWQ on the majority of models. The remaining cases where AWQ leads are attributable to the coarse selection granularity: with only one codebook choice per 128 weights, the method cannot adapt to local variations within each group. With S=16, adding only 0.0625 bpw, AAAC consistently outperforms AWQ on all models.

Comparison with GPTQ. Table 3 compares AAAC against GPTQ on LLaMA-1 and LLaMA-2 models. GPTQ numbers are taken from Lin et al. (2024). AAAC at S=16 outperforms GPTQ on all four models. Even at S=128 (zero overhead), AAAC matches or outperforms GPTQ on all four models.

4.3 Comparison with Gradient-Assisted Methods

As discussed in Section 2, gradient-assisted PTQ methods achieve strong results but at significant cost: OmniQuant requires 1–16 hours of block-wise backpropagation, SqueezeLLM requires 2–4× model size in peak memory for Fisher computation, GPTVQ requires 0.5–11 hours with gradient-based codebook refinement, and QuIP#-FT requires up to 100 GPU-hours of end-to-end fine-tuning.

Table 4 compares AAAC against all four methods. AAAC at S=16 matches or outperforms every method on the majority of models, achieving 60.0% average recovery—matching QuIP#-FT and exceeding OmniQuant (55.0%), SqueezeLLM (55.0%), GPTVQ (25.0%), and QuIP# without fine-tuning (30.0%)—while completing in 3–30 minutes with peak memory no larger than the original model size.

5 Ablation Studies

The goal of this section is to address several questions about the design and applicability of AAAC. First, does AAAC require 16-bit activations at inference time, or can the same quantized model be used with lower-precision activations? We study this in Section 5.2. Second, can AAAC be composed with orthogonal quantization methods? We investigate this in Section 5.1. Third, how sensitive is AAAC

Table 4: g=128 W4A16 WikiText-2 perplexity (↓) vs. gradient-assisted methods. AAAC uses S=16. Best quantized result per row in **bold**.

Model	FP16	RTN	OmniQ	SqLLM	GPTVQ	QuIP#	QuIP#-FT	AAAC
LLaMA-1-7B	5.68	5.96	5.77	5.77	5.96	5.83	**5.76**	**5.76**
LLaMA-1-13B	5.09	5.25	**5.17**	**5.17**	**5.15**	5.20	**5.17**	**5.17**
LLaMA-2-7B	5.47	5.72	5.58	5.57	5.62	5.66	**5.56**	5.58
LLaMA-2-13B	4.88	4.98	4.95	4.96	4.97	5.00	4.95	**4.94**
Average	*5.28*	*5.48*	*5.37*	*5.37*	*5.43*	*5.42*	*5.36*	***5.36***
Recovery	—	—	*55.0%*	*55.0%*	*25.0%*	*30.0%*	*60.0%*	***60.0%***

Table 5: Combining AWQ and AAAC on WikiText-2 (↓). AAAC uses S=g (zero overhead). Best quantized result per row in **bold**.

g	Model	Full	RTN	AWQ	AAAC	AWQ+AAAC
128	LLaMA-1-7B	5.68	5.96	5.81	5.82	**5.80**
128	LLaMA-1-13B	5.09	5.25	5.20	5.21	**5.18**
128	LLaMA-2-7B	5.47	5.72	**5.62**	**5.62**	**5.62**
128	LLaMA-2-13B	4.88	4.98	**4.97**	4.98	4.98
16	Qwen3.5-4B	9.42	10.25	10.16	9.92	**9.52**
16	Qwen3.5-9B	8.51	9.11	8.91	**8.56**	8.67
16	Qwen3.5-27B	6.80	7.02	7.03	6.88	**6.80**
Average		*6.55*	*6.90*	*6.81*	*6.71*	***6.65***
Recovery		—	—	*24.2%*	*53.3%*	***70.5%***

to its hyperparameters? We analyze this in Section 5.3. Finally, how much of AAAC's improvement comes from selecting between multiple codebooks, versus learning those codebooks? The comparison with IF4 at g=16 directly addresses this question. IF4 selects between two codebooks—INT4 and NVFP4 codebooks—but both are fixed, whereas AAAC learns its codebooks. As shown in Table 1, AAAC consistently outperforms IF4 across all evaluated models, indicating that the gains are not explained by codebook selection alone and that learning the codebooks from activation-weighted clustering provides a substantial additional benefit.

5.1 Combining AAAC with AWQ

AWQ modifies the weights before quantization via channel scaling, while AAAC learns the codebook used during quantization. Since they operate on different components, the two methods can be combined: AWQ first scales the weights, then AAAC learns codebooks on the scaled distribution. Table 5 shows that AWQ+AAAC recovers 70.5% of the quantization gap on average, compared to 53.3% for AAAC alone.

5.2 W4A8 Inference

Although AAAC uses activations to weight importance during calibration, it is not sensitive to activation precision at inference time. The same quantized model can serve both W4A16 and W4A8 inference without re-calibration. We verify this in Appendix C, where the same g=128 models are evaluated with FP8 activations; AAAC's improvements carry over fully.

5.3 Sensitivity to Hyperparameters

AAAC has few tunable factors: the selection group size S, the iteration budget, and the calibration set size. At g=16, S=g=16 is the natural choice at zero overhead. At g=128, S=128 (zero overhead) already outperforms AWQ on the majority of models (Tables 2–3), while reducing S to 16 (+0.0625 bpw) consistently improves further. The algorithm is largely insensitive to the iteration budget: a fast configuration (n_{outer}=3, n_{inner}=10, under 2 minutes on an H100 for a 7B model) matches or trails a larger configuration (n_{outer}=10, n_{inner}=30) by only 0.01 perplexity.

Table 6: g=16 W4A16 downstream task accuracy (%↑) on Qwen3.5-9B and Qwen3.5-27B. All methods use the NVFP4 memory layout. Best quantized average in **bold**.

Model	Task	BF16	RTN	IF4	AWQ	AAAC
Qwen3.5-9B	HellaSwag	78.1	77.3	77.3	77.3	77.6
	ARC-C	56.0	55.3	55.5	53.8	54.9
	PIQA	79.9	79.2	80.0	80.0	80.0
	BBH	62.3	60.6	60.1	61.4	61.7
	MuSR	43.4	42.1	40.5	44.2	44.2
	GPQA	42.2	39.7	41.9	41.4	40.0
	MATH	47.9	47.4	49.0	48.6	48.9
	MMLU	78.7	77.3	77.3	77.3	78.1
Qwen3.5-27B	HellaSwag	83.4	83.0	83.2	83.2	83.2
	ARC-C	61.7	61.9	61.3	61.9	61.9
	PIQA	82.1	82.3	82.2	81.8	82.5
	BBH	73.1	72.5	72.2	72.8	72.3
	MuSR	52.5	53.7	51.3	52.0	51.6
	GPQA	50.1	49.3	48.2	49.3	48.5
	MATH	57.6	55.7	57.4	56.1	57.4
	MMLU	84.5	84.2	84.5	84.2	84.7
Average		*64.59*	*63.84*	*63.87*	*64.08*	***64.21***
Recovery		—	—	*4.0%*	*32.0%*	***49.3%***

Calibration data requirements are similarly minimal: in our experience, 1 sequence (2,048 tokens) yields perplexity within 0.01 of the result with 4 or more sequences. We use 4 sequences as a conservative default.

6 Downstream Task Evaluation

To complement the perplexity results, we evaluate on 8 downstream tasks using Qwen3.5-9B and Qwen3.5-27B at g=16. All baselines (IF4 and AWQ) use the NVFP4 memory layout at this group size. Tasks span reasoning (BBH, MuSR), knowledge (MMLU, GPQA), commonsense (HellaSwag, ARC-C, PIQA), and mathematics (MATH). All evaluations use the default few-shot settings from lm-evaluation-harness. Table 6 reports accuracy on each task. AAAC achieves the highest average accuracy among all quantized methods (64.21%), recovering most of the BF16 baseline (64.59%).

7 Limitations

Our evaluation covers models up to 32B parameters; we have not tested on 70B+ models where quantization dynamics may differ. The comparison with gradient-assisted methods is limited to four LLaMA models (7B and 13B), as these are the models for which prior work reports numbers under comparable evaluation protocols. Finally, AAAC targets weight-only quantization (W4A16 and W4A8) and does not address the joint weight-activation setting (W4A4).

8 Conclusion

We have presented AAAC (Activation-Aware Adaptive Codebooks), a lightweight post-training quantization method that learns two small scalar codebooks per layer (64 bytes) via activation-weighted k-means on a small calibration set. Despite this simplicity, AAAC consistently outperforms all gradient-free baselines and matches gradient-assisted methods that require hours of optimization and higher memory.

THE USE OF LARGE LANGUAGE MODELS (LLMS)

In this work, we used generative AI tools for research ideation and execution. Specifically, we used LLMs to implement parts of the quantization and calibration code. All LLM-assisted design decisions were reviewed by the authors, and all LLM-generated code was inspected, tested, and verified against reference implementations and other paper results by authors before being used to produce any reported result. We have not used generative AI tools for generating synthetic datasets, developing theoretical models, formulating or proving mathematical claims, translation, dataset cleaning or reformatting, or qualitative and thematic data analysis; the remaining categories requiring disclosure are not applicable to this work.

Additionally, we used generative AI tools for retrieval and discovery of related work, for drafting sections of the paper, and for editing the manuscript to improve readability. Literature identified with LLM assistance was verified by the authors by reading the primary sources directly, and all numerical claims attributed to prior work were checked against the original papers. All drafted and edited text was reviewed and revised by the authors.

We have reviewed all AI-assisted work, and we take responsibility for the final content of this work, including text, claims, or artifacts produced with the aid of generative AI.

REFERENCES

Jack Cook, Hyemin S. Lee, Kathryn Le, Junxian Guo, Giovanni Traverso, Anantha P. Chandrakasan, and Song Han. Adaptive block-scaled data types, 2026. URL https://arxiv.org/abs/2603.28765.

Tim Dettmers, Ruslan Svirschevski, Vage Egiazarian, Denis Kuznedelev, Elias Frantar, Saleh Ashkboos, Alexander Borzunov, Torsten Hoefler, and Dan Alistarh. Spqr: A sparse-quantized representation for near-lossless llm weight compression, 2023. URL https://arxiv.org/abs/2306.03078.

Elias Frantar, Saleh Ashkboos, Torsten Hoefler, and Dan Alistarh. Gptq: Accurate post-training quantization for generative pre-trained transformers, 2023. URL https://arxiv.org/abs/2210.17323.

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.

Aaron Grattafiori, Abhimanyu Dubey, Abhinav Jauhri, Abhinav Pandey, Abhishek Kadian, Ahmad Al-Dahle, Aiesha Letman, Akhil Mathur, Alan Schelten, Alex Vaughan, Amy Yang, Angela Fan, Anirudh Goyal, Anthony Hartshorn, Aobo Yang, Archi Mitra, Archie Sravankumar, Artem Korenev, Arthur Hinsvark, Arun Rao, Aston Zhang, Aurelien Rodriguez, Austen Gregerson, Ava Spataru, Baptiste Roziere, Bethany Biron, Binh Tang, Bobbie Chern, Charlotte Caucheteux, Chaya Nayak, Chloe Bi, Chris Marra, Chris McConnell, Christian Keller, Christophe Touret, Chunyang Wu, Corinne Wong, Cristian Canton Ferrer, Cyrus Nikolaidis, Damien Allonsius, Daniel Song, Danielle Pintz, Danny Livshits, Danny Wyatt, David Esiobu, Dhruv Choudhary, Dhruv Mahajan, Diego Garcia-Olano, Diego Perino, Dieuwke Hupkes, Egor Lakomkin, Ehab AlBadawy, Elina Lobanova, Emily Dinan, Eric Michael Smith, Filip Radenovic, Francisco Guzmán, Frank Zhang, Gabriel Synnaeve, Gabrielle Lee, Georgia Lewis Anderson, Govind Thattai, Graeme Nail, Gregoire Mialon, Guan Pang, Guillem Cucurell, Hailey Nguyen, Hannah Korevaar, Hu Xu, Hugo Touvron, Iliyan Zarov, Imanol Arrieta Ibarra, Isabel Kloumann, Ishan Misra, Ivan Evtimov, Jack Zhang, Jade Copet, Jaewon Lee, Jan Geffert, Jana Vranes, Jason Park, Jay Mahadeokar, Jeet Shah, Jelmer van der Linde, Jennifer Billock, Jenny Hong, Jenya Lee, Jeremy Fu, Jianfeng Chi, Jianyu Huang, Jiawen Liu, Jie Wang, Jiecao Yu, Joanna Bitton, Joe Spisak, Jongsoo Park, Joseph Rocca, Joshua Johnstun, Joshua Saxe, Junteng Jia, Kalyan Vasuden Alwala, Karthik Prasad, Kartikeya Upasani, Kate Plawiak, Ke Li, Kenneth Heafield, Kevin Stone, Khalid El-Arini, Krithika Iyer, Kshitiz Malik, Kuenley Chiu, Kunal Bhalla, Kushal Lakhotia, Lauren Rantala-Yeary, Laurens van der Maaten, Lawrence Chen, Liang Tan, Liz Jenkins, Louis Martin, Lovish Madaan, Lubo Malo,

Lukas Blecher, Lukas Landzaat, Luke de Oliveira, Madeline Muzzi, Mahesh Pasupuleti, Mannat Singh, Manohar Paluri, Marcin Kardas, Maria Tsimpoukelli, Mathew Oldham, Mathieu Rita, Maya Pavlova, Melanie Kambadur, Mike Lewis, Min Si, Mitesh Kumar Singh, Mona Hassan, Naman Goyal, Narjes Torabi, Nikolay Bashlykov, Nikolay Bogoychev, Niladri Chatterji, Ning Zhang, Olivier Duchenne, Onur Çelebi, Patrick Alrassy, Pengchuan Zhang, Pengwei Li, Petar Vasic, Peter Weng, Prajjwal Bhargava, Pratik Dubal, Praveen Krishnan, Punit Singh Koura, Puxin Xu, Qing He, Qingxiao Dong, Ragavan Srinivasan, Raj Ganapathy, Ramon Calderer, Ricardo Silveira Cabral, Robert Stojnic, Roberta Raileanu, Rohan Maheswari, Rohit Girdhar, Rohit Patel, Romain Sauvestre, Ronnie Polidoro, Roshan Sumbaly, Ross Taylor, Ruan Silva, Rui Hou, Rui Wang, Saghar Hosseini, Sahana Chennabasappa, Sanjay Singh, Sean Bell, Seohyun Sonia Kim, Sergey Edunov, Shaoliang Nie, Sharan Narang, Sharath Raparthy, Sheng Shen, Shengye Wan, Shruti Bhosale, Shun Zhang, Simon Vandenhende, Soumya Batra, Spencer Whitman, Sten Sootla, Stephane Collot, Suchin Gururangan, Sydney Borodinsky, Tamar Herman, Tara Fowler, Tarek Sheasha, Thomas Georgiou, Thomas Scialom, Tobias Speckbacher, Todor Mihaylov, Tong Xiao, Ujjwal Karn, Vedanuj Goswami, Vibhor Gupta, Vignesh Ramanathan, Viktor Kerkez, Vincent Gonguet, Virginie Do, Vish Vogeti, Vítor Albiero, Vladan Petrovic, Weiwei Chu, Wenhan Xiong, Wenyin Fu, Whitney Meers, Xavier Martinet, Xiaodong Wang, Xiaofang Wang, Xiaoqing Ellen Tan, Xide Xia, Xinfeng Xie, Xuchao Jia, Xuewei Wang, Yaelle Goldschlag, Yashesh Gaur, Yasmine Babaei, Yi Wen, Yiwen Song, Yuchen Zhang, Yue Li, Yuning Mao, Zacharie Delpierre Coudert, Zheng Yan, Zhengxing Chen, and Zoe Papakipos. The llama 3 herd of models, 2024. URL https://arxiv.org/abs/2407.21783.

Sehoon Kim, Coleman Hooper, Amir Gholami, Zhen Dong, Xiuyu Li, Sheng Shen, Michael W. Mahoney, and Kurt Keutzer. Squeezellm: Dense-and-sparse quantization, 2024. URL https://arxiv.org/abs/2306.07629.

Ji Lin, Jiaming Tang, Haotian Tang, Shang Yang, Wei-Ming Chen, Wei-Chen Wang, Guangxuan Xiao, Xingyu Dang, Chuang Gan, and Song Han. Awq: Activation-aware weight quantization for llm compression and acceleration, 2024. URL https://arxiv.org/abs/2306.00978.

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

Qwen, :, 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.

Colin Raffel, Noam Shazeer, Adam Roberts, Katherine Lee, Sharan Narang, Michael Matena, Yanqi Zhou, Wei Li, and Peter J. Liu. Exploring the limits of transfer learning with a unified text-to-text transformer, 2023. URL https://arxiv.org/abs/1910.10683.

Wenqi Shao, Mengzhao Chen, Zhaoyang Zhang, Peng Xu, Lirui Zhao, Zhiqian Li, Kaipeng Zhang, Peng Gao, Yu Qiao, and Ping Luo. Omniquant: Omnidirectionally calibrated quantization for large language models, 2024. URL https://arxiv.org/abs/2308.13137.

Hugo Touvron, Thibaut Lavril, Gautier Izacard, Xavier Martinet, Marie-Anne Lachaux, Timothée Lacroix, Baptiste Rozière, Naman Goyal, Eric Hambro, Faisal Azhar, Aurelien Rodriguez, Armand Joulin, Edouard Grave, and Guillaume Lample. Llama: Open and efficient foundation language models, 2023. URL https://arxiv.org/abs/2302.13971.

Albert Tseng, Jerry Chee, Qingyao Sun, Volodymyr Kuleshov, and Christopher De Sa. Quip#: Even better llm quantization with hadamard incoherence and lattice codebooks, 2024. URL https://arxiv.org/abs/2402.04396.

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, 2025. URL https://arxiv.org/abs/2402.15319.

A g=16 RESULTS ON QWEN2.5

Table 7 presents g=16 WikiText-2 results on the Qwen2.5 family. AAAC achieves the best result on all four models.

Table 7: g=16 W4A16 perplexity on Qwen2.5 models, WikiText-2 (↓). Best quantized result in **bold**.

Model	BF16	RTN	IF4	AWQ	AAAC
Qwen2.5-0.5B	13.03	14.33	14.11	14.17	**13.66**
Qwen2.5-7B	6.80	7.03	6.97	7.00	**6.93**
Qwen2.5-14B	5.25	5.59	5.54	5.54	**5.45**
Qwen2.5-32B	4.97	5.17	5.12	5.15	**5.09**
Average	*7.51*	*8.03*	*7.94*	*7.97*	***7.78***
Recovery	—	—	*18.4%*	*12.6%*	***47.8%***

Table 8 presents the corresponding C4 results. The same pattern holds across all models.

Table 8: g=16 W4A16 perplexity on Qwen2.5 models, C4 (↓). Best quantized result in **bold**.

Model	BF16	RTN	IF4	AWQ	AAAC
Qwen2.5-0.5B	20.02	21.82	21.57	21.54	**20.95**
Qwen2.5-7B	11.77	12.06	12.02	11.99	**11.93**
Qwen2.5-14B	10.25	10.52	10.49	10.45	**10.39**
Qwen2.5-32B	10.09	10.23	10.21	10.21	**10.18**
Average	*13.03*	*13.66*	*13.57*	*13.55*	***13.36***
Recovery	—	—	*13.6%*	*17.6%*	***47.2%***

B g=16 C4 RESULTS

Table 9 presents C4 perplexity for the same models as Table 1. AAAC achieves the lowest perplexity on all models, with 50.9% average gap recovery.

Table 9: g=16 W4A16 perplexity on C4 (↓). AAAC uses S=g=16 (zero overhead). Best quantized result in **bold**.

Model	BF16	RTN	IF4	AWQ	AAAC
Llama-3.2-1B	13.79	15.60	15.30	15.15	**14.62**
Llama-3.2-3B	11.20	12.02	11.86	11.81	**11.58**
Llama-3.1-8B	9.50	10.21	10.07	10.07	**9.84**
Qwen3.5-2B	17.58	18.59	18.40	18.29	**18.13**
Qwen3.5-4B	13.99	14.51	14.37	14.38	**14.28**
Qwen3.5-9B	12.19	12.57	12.53	12.44	**12.38**
Qwen3.5-27B	10.47	10.64	10.61	10.59	**10.55**
Average	*12.67*	*13.45*	*13.31*	*13.25*	***13.05***
Recovery	—	—	*18.5%*	*26.0%*	***50.9%***

C W4A8 RESULTS

Although AAAC uses activations to weight importance during calibration, it is not sensitive to activation precision at inference time. The same g=128 quantized models from Table 2 are evaluated

with per-tensor FP8 activation quantization. Table 10 confirms that AAAC's improvements carry over to the W4A8 setting, with AAAC at S=16 recovering 49.1% of the quantization gap.

Table 10: g=128 W4A8 WikiText-2 perplexity (↓) vs. AWQ. Same quantized models as Table 2 with FP8 activations. **Bold**: best result per model.

				AAAC	
Model	FP16	RTN	AWQ	S=128	S=16
Llama-3.2-3B	7.85	8.54	8.26	**8.20**	**8.15**
Llama-3.1-8B	6.27	6.87	6.65	6.72	**6.60**
Qwen2.5-0.5B	13.22	15.74	15.05	**14.64**	**14.44**
Qwen2.5-7B	6.89	7.27	7.11	7.17	**7.11**
Qwen2.5-14B	5.33	5.82	5.70	**5.69**	**5.64**
Average	*7.91*	*8.85*	*8.55*	*8.48*	***8.39***
Recovery	—	—	*31.4%*	*38.9%*	***49.1%***