Calibration and Transformation-Free Weight-Only LLMs Quantization via Dynamic Grouping

Anonymous ACL submission

Abstract

Large Language Models (LLMs) deliver strong performance but are difficult to deploy under tight memory and compute constraints. Low-bit post-training quantization (PTQ) is a promising direction; however, it typically relies on calibration data, auxiliary transformations, and GPU tools. To address these limitations, we propose Multi Scale Binary (MSB), a calibration-free and transformation-free PTQ method that generalizes binary quantization to multi-bit settings. MSB optimizes a variance-based dynamic grouping objective that assigns shared scales to groups of weights, yielding a multi-scale binary representation. This formulation applies consistently across quantization granularities, from per-tensor to block-wise settings, without requiring calibration data or intermediate transformations. We implement the optimization in a CPU based solver for the quantization step and evaluate using standard bfloat16 execution without low-bit packing. On Llama 3.2 3B, MSB achieves 8.43 perplexity on WikiText-2 under 4-bit weight only block-wise quantization, compared to 7.81 in full precision and 12.23 with GPTQ in its default setup. Overall, MSB provides a new optimization perspective for low-bit PTQ while simplifying the pipeline by removing calibration and transformations.

1 Introduction

Large Language Models (LLMs) such as LLaMA (Dubey et al., 2024), DeepSeek (Guo et al., 2025), and Gemma (Gemma-Team et al., 2025) achieve state-of-the-art (SOTA) performance across a wide range of Natural Language Processing (NLP) tasks, but their parameter scale creates substantial deployment costs. Even under weight-only storage, modern models can require hundreds of gigabytes of memory (Guo et al., 2025), motivating compression techniques that reduce the footprint of pretrained weights.

Quantization addresses this challenge by representing weights and optionally activations with fewer bits, reducing memory traffic and enabling low precision arithmetic (Dettmers et al., 2022). As a result, quantization is central to deployment in both cloud scale serving and edge/on-device environments (Tan et al., 2024). In particular, we focus on weight-only post-training quantization (PTQ) in low-bit regimes. While 4-bit quantization is widely studied due to its strong compression, slightly higher bit-widths such as 6-bit per-tensor quantization often exhibit improved robustness, motivating methods that generalize across granularities.

Existing low-bit quantization approaches for LLMs include post-training quantization (PTQ), which produces a quantized model directly from pretrained weights (Frantar et al., 2022). PTQ is particularly practical when compute budgets, time constraints, or data access limitations preclude retraining or quantization-aware training; we therefore focus on weight-only PTQ under low-bit regimes.

Within this setting, we consider two representative configurations: 4-bit block-wise quantization for efficient compression (Frantar et al., 2022; Liu et al., 2025) and 6-bit per-tensor quantization for improved robustness, and evaluate whether a unified framework can support both regimes. We do not target ultra low bit widths, which often constitute a qualitatively different regime. In this 4-bit setting, practical pipelines often pair the quantizer with calibration and/or function preserving reparameterizations, such as rescaling or rotations, to mitigate outliers and reduce quantization error (Lin et al., 2023; Liu et al., 2025). While effective, these ingredients introduce extra dependencies and design choices. In this work, we study a complementary setting: calibration-free and transformation-free PTQ that operates solely on pretrained weights. This formulation removes dependencies on calibra-

tion data and reparameterization steps, enabling a simpler objective-driven view of low-bit quantization.

Our design is motivated by binary PTQ, where uniform binarization is typically insufficient for LLMs. Effective methods therefore rely on structured partitioning: they reserve limited higher capacity representations (e.g., residual or mixed precision) for the most sensitive weights while binarizing the remainder (Huang et al., 2024). This suggests a broader principle that under tight precision budgets, performance depends not only on the quantizer, but also on how weights are partitioned and how representational capacity is allocated across groups of heterogeneous sensitivity. Building on this principle, we ask whether it can

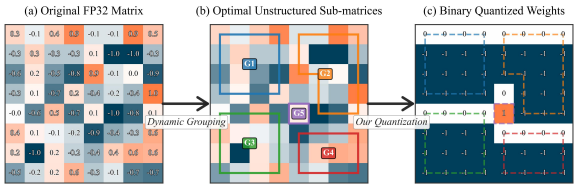

Figure 1: Overview of our dynamic grouping framework for calibration-free, transformation-free low-bit PTQ.

generalize to practical n-bit quantization beyond 1-bit. Instead of explicitly selecting salient weights, we allocate representational capacity through optimized grouping, by introducing a multi scale binary (MSB) approximation: different groups of weights share different scale parameters while retaining a binary sign structure. We formulate an MSB objective and optimize it with variance-aware magnitude grouping using greedy solvers, yielding a calibration-free and transformation-free framework that applies to both whole-matrix and block-wise settings.

In this work, we (i) formulate a calibration-free, transformation-free objective for MSB weight approximation and derive an equivalent magnitude grouping form based on within group statistics, yielding a variance based grouping objective; (ii) develop objective driven, distribution sensitive solvers, including a dynamic programming reference and efficient approximations, Greedy Grouping and Windowed Greedy Merging (WGM), implemented for CPU-based offline PTQ; and (iii) demonstrate that a single objective–solver framework supports both per-tensor (6-bit) and block-wise (4-bit) quantization, achieving competitive accuracy without calibration or transformation steps.

2 Related Work

2.1 Scalar Quantization and Clustering

Quantization is closely related to scalar quantization, where values are partitioned to minimize distortion (Lloyd, 1982; Max, 1960). Applying k-means to magnitudes is a natural interpretation of such grouping (McQueen, 1967). Our approach shares this connection but derives the grouping criterion from a weight-only PTQ objective with additional structural constraints tailored to low-bit LLM quantization.

2.2 PTQ for LLMs

Rounding based weight-only baselines. A common starting point for PTQ is round-to-nearest (RTN), which applies uniform quantization using per-tensor or block-wise scaling (and optionally a zero point or clipping) and rounds each weight to the nearest quantization level. Practical toolchains, e.g. bitsandbytes (BnB) (Dettmers et al., 2022, 2023), further adopt block-wise scaling and packed low-bit formats, including widely used 4-bit representations such as FP4/NF4, to reduce memory footprint and improve throughput.

Calibration-free optimization. Beyond direct rounding, calibration-free methods optimize quantized weights using only pretrained parameters. For example, HQQ (Badri and Shaji, 2023) formulates weight-only quantization as a robust optimization problem and solves it efficiently (e.g., via half-quadratic splitting) without calibration data, aiming to reduce distortion compared with naive rounding while keeping a simple inference time representation.

Calibration-based error minimization and second-order variants. Methods such as GPTQ (Frantar et al., 2022) improve weight-only PTQ by explicitly minimizing output error under calibration. A key mechanism is to use a small calibration set to estimate layer-wise sensitivity via approximate second-order information and then perform sequential/block-wise quantization with error compensation, substantially reducing accuracy loss at 3 and 4 bits compared with pure rounding.

Calibration- and transformation-based PTQ. Another widely used line of work combines low-bit quantization with lightweight function-preserving reparameterizations to mitigate outliers and improve quantizability. Representative examples include equivalent rescaling/shifting based on activation statistics (e.g., AWQ (Lin et al., 2023)), dif-

ferentiable tuning of clipping and equivalent transformations (e.g., OmniQuant (Shao et al., 2023)), and rotation-based reparameterizations that reduce outliers prior to quantization (e.g., SpinQuant (Liu et al., 2025)). These approaches can be highly effective, but they introduce additional data dependencies for calibration and extra design choices for transformation parameterizations and tuning.

2.3 Binary PTQ for LLMs

Traditional binary quantization such as XNOR-Net (Rastegari et al., 2016) represents weights with 1-bit values and is often characterized by an objective that approximates real-valued weights with a scaled binary form. Modern binary PTQ for LLMs like BiLLM (Huang et al., 2024) rarely binarizes all weights uniformly; instead, effective schemes apply heterogeneous treatment, allocate limited higher capacity representations (e.g., residual terms) to sensitive weights while binarizing the remaining majority, preventing severe degradation. This regime highlights a core principle: under extreme precision constraints, performance depends not only on the quantizer, but also on how weights are partitioned and how limited representational capacity is allocated across groups of heterogeneous sensitivity.

Connection to our work. Motivated by the objective-centric view and heterogeneous treatment in binary PTQ, we generalize this perspective to MSB PTQ. Our method studies a complementary setting that operates directly on pretrained weights, avoids calibration data and function-preserving transformations, and optimizes an n-bit objective via distribution-sensitive grouping.

3 Methodology

This section formalizes our dynamic grouping objective and presents three solvers spanning accuracy runtime trade-offs. We describe the boundary parameter λ and analyze the time/memory costs of each variant.

3.1 Preliminary

The classical objective for binary quantization of XNOR-Net (Rastegari et al., 2016) approximates a full precision weight tensor by a scaled binary codebook. Given a weight matrix $\mathbf{W} \in \mathbb{R}^{m\times l}$, XNOR-style binarization seeks a binary matrix $\mathbf{B} \in \{-1,+1\}^{m\times l}$ and a positive scale $\alpha > 0$ such that

$$\alpha^*, \boldsymbol{B}^* = \arg\min_{\alpha,\boldsymbol{B}} \|\boldsymbol{W} - \alpha\boldsymbol{B}\|_2^2 . \qquad (1)$$

This problem admits a closed form solution: $\mathbf{B}^\star = \mathrm{sign}(\mathbf{W})$ and $\alpha^\star = \|\mathbf{W}\|_1/(ml)$ (Rastegari et al., 2016).

This formulation captures the canonical scaled binary form $\{\pm\alpha\}$ and provides an objective-centric viewpoint that we later generalize to multi-bit settings.

3.2 MSB Optimization Objective

Based on the ideas discussed above, we propose a new MSB optimization objective that minimizes the total quantization loss 1 across all unstructured sub-matrices by identifying their optimal grouping based on a predetermined quantization loss.

The following variables define our optimization objective:

- Let $\boldsymbol{A} \in \mathbb{R}^{m\times n}$ be the given matrix and $\mathbb{A}$ be the all possible set of arbitrary grouping of $\boldsymbol{A}$.
- g represents the number of non-overlapping unstructured sub-matrices, satisfying $1 \le g \le |\boldsymbol{A}|$, where $|\boldsymbol{A}|$ denotes the total number of non-zero elements in matrix $\boldsymbol{A}$.
- $\alpha \in \mathbb{R}$ and $\boldsymbol{B} \in \{-1,1\}^{m\times n}$ are used for the binary approximation of $\boldsymbol{A}$. Each α_i and $\boldsymbol{B}_i$ approximates one $\boldsymbol{A}_i$, so $\boldsymbol{B}_i$ refers to the same set of matrix elements as partition $\boldsymbol{A}_i$.

Thus, we partition $\boldsymbol{B}$ identically to $\boldsymbol{A}$, such that $\boldsymbol{A}_1 = \boldsymbol{B}_1, \ldots, \boldsymbol{A}_g = \boldsymbol{B}_g$ and $\mathbb{A} = \mathbb{B}$ in turns of matrix index. We group matrix $\boldsymbol{A}$ into g non-overlapping sub-matrices $\{\boldsymbol{A}_i\}_{i=1}^g = \{\boldsymbol{A}_1, \boldsymbol{A}_2, \ldots, \boldsymbol{A}_g\} \in \mathbb{A}$. For simplicity, we focus on non-zero elements and place any zero elements into a special group, if they exist. This special group is omitted from consideration as it has a zero loss. Additionally, $\mathbb{A}$ encompasses all possible non-overlapping partitions using all elements of $\boldsymbol{A}$. Based on this, our proposed optimization objective is:

$$g^*, \{\boldsymbol{A}_i^*\}_{i=1}^{g^*} = \arg\min_{\substack{\{\boldsymbol{A}_i\}_{i=1}^g \in \mathbb{A} \\ \{\boldsymbol{B}_i\}_{i=1}^g \in \mathbb{B} \\ 1\le g\le|\boldsymbol{A}|}} \mathcal{L}(\{\boldsymbol{A}_i, \boldsymbol{B}_i\}_{i=1}^g)$$

$$\mathcal{L}(\{\boldsymbol{A}_i, \boldsymbol{B}_i\}_{i=1}^g) = \sum_{i=1}^{g} ||\boldsymbol{A}_i - \alpha_i\boldsymbol{B}_i||_2^2 + \frac{1}{|\boldsymbol{B}_i|}.$$

The optimization objective consists of two main components. The first summation represents the quantization loss. The second summation serves as a regularization term that penalizes the inverse of the group size. We penalize the inverse of the group size because smaller groups lead to more partitions, resulting in lower efficiency due to a smaller group for each scale term α. It is important to note that the partitions need not be block-structured; each partition is simply a set of elements from $\boldsymbol{A}$. When combining all g sub-matrices, we recover the original matrix $\boldsymbol{A}$. To solve the objective, by substituting the optimal values, we can rewrite the squared 2-norm.

$$||\boldsymbol{A} - \alpha^* \boldsymbol{B}^*||_2^2 = ||\boldsymbol{A}||_2^2 - \frac{||\boldsymbol{A}||_1^2}{|\boldsymbol{A}|}.$$

Consider the variance of absolute value of $\boldsymbol{A}_i$, let $\tilde{\boldsymbol{A}}_i$ be $\boldsymbol{A}_i$ with element-wise absolute value,

$$\mathrm{Var}(\tilde{\boldsymbol{A}}_i) = \frac{||\boldsymbol{A}_i||_2^2}{|\boldsymbol{A}_i|} - \left(\frac{||\boldsymbol{A}_i||_1}{|\boldsymbol{A}_i|}\right)^2$$
$$|\boldsymbol{A}_i|\mathrm{Var}(\tilde{\boldsymbol{A}}_i) = ||\boldsymbol{A}_i - \alpha_i^* \boldsymbol{B}_i^*||_2^2.$$

Finally, since $|\mathbf{A}_i| = |\mathbf{B}_i|$, consider grouping $\mathcal{G} = \{\mathbf{A}_i\}_{i=1}^g \in \mathbb{A}$.

$$\mathrm{cost}(\mathcal{G}) = \sum_{i=1}^{g} \left(|\mathbf{A}_i| \, \mathrm{Var}(\tilde{\mathbf{A}}_i) + \frac{1}{|\mathbf{A}_i|}\right).$$

We rewrite the objective:

$$g^*, \mathcal{G}^* = \underset{\mathcal{G}\in\mathbb{A},\, 1\le g\le|\mathbf{A}|}{\arg\min} \; \mathrm{cost}(\mathcal{G}). \tag{2}$$

3.3 Three Tailored Algorithms

To solve the optimization objective, we propose three algorithms, each tailored to different accuracy runtime regimes. Algorithm 1, Dynamic Grouping (DG), employs classic dynamic programming to guarantee optimal solutions. Meanwhile, Algorithm 2, Greedy Grouping (GG), adopts a heuristic strategy to improve computational efficiency with reasonable solution quality. Algorithm 3, Windowed Greedy Merging (WGM), is an efficient approximation method based on Greedy Grouping that achieves a strong balance between quantization performance and speed. A local refinement variant based on WGM (denoted WGM-L) and detailed pseudo code for Algorithm 1, 2 and WGM-L is given in Appendix A.

3.3.1 Algorithm 1, Dynamic Grouping

Dynamic Grouping (DG) solves the optimization objective via dynamic programming by partitioning the sorted sequence of magnitudes into contiguous intervals. The optimal substructure follows a standard segmentation formulation, where the optimal solution for n elements and k groups is obtained by selecting the best split point i that minimizes

$$dp[k][n] = \min_{1\le i<n} dp[k-1][i] + f([i:n]). \tag{3}$$

Here f denotes the per-group cost defined in Eq. (2). Prefix sums enable constant-time evaluation of interval costs, making the formulation efficient despite its cubic worst-case complexity. Full derivation and implementation details are provided in Appendix A.

3.3.2 Algorithm 2, Greedy Grouping

Modern LLMs often have linear layers with about 10^6 weights per matrix (e.g., 1024×1024) or more, making exact dynamic programming impractical; we therefore adopt a greedy merging heuristic, which is fast enough to be feasible at this scale. We therefore approximate Algorithm 1 with a greedy merging heuristic: we sort nonzero entries, initialize singleton groups, and maintain a min-heap of adjacent merge costs (size $mn-1$). We repeatedly pop the minimum-cost merge, update its neighboring costs, and stop once the target number of groups is reached.

3.3.3 Algorithm 3, Windowed Greedy Merging

To improve Algorithm 1, and achieve a better efficiency-quality trade-off, we modify the initial merge cost array, and propose our Algorithm 3. Instead of starting at mn groups each of size 1, we start at mn/k groups, where k is the window size for each initial group. We provide a detailed description of the algorithm 3 below.

Overall, DG provides an optimal reference solution, GG offers a simple and scalable approximation, and WGM achieves an effective balance between quantization performance and efficiency, particularly in large-scale settings. These variants form a practical solver family for the underlying optimization objective.

3.4 Interpretation and Boundary of λ

We introduce a regularization weight λ to trade off the variance term against a size-dependent penalty

Algorithm 3 Windowed Greedy Merging (WGM)

WGM: $\boldsymbol{A} \in \mathbb{R}^{m\times n}, g, k$
sort absolute value of non-zero entry into an array
initial the merge cost array $[(i$ (group start), $i+k$(group end), $cost, \mu)]$ for i in range(0, mn - 1, k)
initialize the ignore array
heapify the merge cost array
while len(merge cost array) $>$ g **do**
  currentMerge $\leftarrow$ heapop
  if currentMerge in ignore array **then**
    remove currentMerge from ignore array
    continue
  end if
  push two new neighbouring merges as updates
  update ignore array by two old merges, invalidated by the two new merges
end while
return merge cost array

in the dynamic grouping objective. Since the penalty scales as $\lambda/|\mathbf{A}_i|$, smaller λ favors finer partitions (more, smaller groups), whereas larger λ encourages coarser groupings (fewer, larger groups). To keep the two terms comparable across group sizes, we normalize the variance contribution by group mass, leading to

$$\text{cost}(\mathcal{G}) \;=\; \sum_{i=1}^{g}\left(\frac{|\boldsymbol{A}_i|}{|\boldsymbol{A}|}Var(\tilde{\boldsymbol{A}}_i) + \frac{\lambda}{|\boldsymbol{A}_i|}\right).$$

Algorithmically, this only changes the per-group cost by applying the above normalization and scaling the penalty by λ. We define $\lambda_{\min}$ and $\lambda_{\max}$ as the values that induce, respectively, the finest admissible partition and the single-group solution, and reparameterize $\lambda = \Lambda(\tilde{\lambda})$ with $\tilde{\lambda} \in [0, 1]$ via a monotone map for interpretability. See Appendix H for additional details and derivation.

3.5 Time Complexity

Given the input matrix $\boldsymbol{A} \in \mathbb{R}^{m\times n}$, the sorting of all non-zero entries takes $O(mn\log(mn))$, prefix sum and prefix squared 2 norm takes $O(mn)$, complete the dynamic programming table takes $O((mn)^2 \cdot \text{max group number})$ since computing cost using prefix sum and prefix squared 2 norm takes $O(1)$. So Dynamic Grouping's time complexity is $O((mn)^3)$, which is infeasible to run to completion.

Then, Greedy Grouping is designed to approximate the optimal solution faster. We sort all non-zero entries and use a min heap storing the merging cost between each group, i.e. heap has size mn. Then, we update the heap until only the desired number of groups remains. This greedy merging only takes $O(mn\log(mn))$ since the sorting of non-zero entries takes $O(mn\log(mn))$, popping and updating the heap takes $O(mn\log(mn))$.

For the WGM, it effectively reduces the merging time complexity to $O(\frac{mn}{k}\log(\frac{mn}{k}))$ at the cost of accuracy, since we may overlook a better merge in the initial mn/k groups of size k.

3.6 Inference Representation

In the current evaluation we use bfloat16 simulation to isolate the grouping objective from backend implementation effects. A packed implementation would store sign bits and scale-group indices, with per-block scale tables. All grouping decisions are precomputed during quantization, so inference consists of static lookup of indices and scales without any dynamic grouping. We do not claim end-to-end speedup in this paper, and optimized CUDA kernels are left to future work.

4 Experiments

We evaluate MSB under two granularities. Our primary setting is 4-bit block-wise quantization, in which each weight matrix is partitioned into 64-row blocks that are quantized independently for efficiency. We also report 6-bit per-tensor results, where each matrix is quantized as a whole using a single global grouping. Algorithm 3 (WGM) is used in both regimes, while Together, the results suggest that our framework is robust to the choice of quantization granularity, performing well in both tensor-level and block-level regimes. Additional experimental details and results for all four algorithms, as well as extended tables for both per-tensor and block-wise settings, are provided in Appendix B.

4.1 Settings

We evaluate under simulated MSB PTQ. For a target bit-width b, each weight in a group is represented as

$$\hat{w} = \alpha_i \cdot s, s \in \{-1, +1\}, \;\; i \in \{1, \ldots, 2^{b-1}\},$$

i.e., a binary sign multiplied by one of 2^{b-1} group specific positive scales $\{\alpha_i\}$. This yields a symmetric 2^b-level codebook $\{\pm\alpha_i\}$ while retaining

a binary sign structure. All quantized values are decoded and stored in `bfloat16`, so our results isolate the effect of the proposed grouping optimization from backend specific implementations. As for storage, for 4-bit block-wise quantization, the theoretical effective storage is 6.00 bits/weight without double quantization (DQ) (Dettmers et al., 2023) or 4.78 bits/weight with DQ. For per-tensor quantization, metadata overhead is negligible relative to the weight tensor size. We report main results without DQ, and Appendix E reports the model-dependent QA and PPL impact of DQ. We adopt simulation because the goal of this work is to validate the proposed optimization objective; implementing a dedicated low-bit kernel is orthogonal and not prioritized given the relatively high time complexity of the quantization procedure. Exact zeros, when present, are kept as `bfloat16` zeros; in practice such entries are extremely sparse and do not affect codebook construction. For our main experiments, we use WGM with window size $w = 64$ for per-tensor quantization and $w = 1$ for block-wise quantization.

4.1.1 Models and datasets

We facilitate our method on the LLama, Falcon (Falcon-LLM-Team, 2024) and Gemma families. We evaluate quantization performance using seven zero-shot common sense QA tasks used in related researches (Shang et al., 2023; Huang et al., 2024), enabling direct comparability with prior research, i.e. ARC-C and ARC-E (Clark et al., 2018), BooIQ (Clark et al., 2019), HellaSwag (Zellers et al., 2019), OPQA (Mihaylov et al., 2018), PIQA (Bisk et al., 2020), WinoGrande (Sakaguchi et al., 2021). Next, we conduct experiments on perplexity (PPL) using three datasets that have also been utilized in previous research (Shang et al., 2023; Huang et al., 2024), i.e. the Wikitext 2 (WK2) (Merity et al., 2016), the PTB (Marcus et al., 1993) and the C4 (Raffel et al., 2020).

4.1.2 Baseline

We choose baselines to reflect the main families of weight-only low-bit PTQ used for LLMs, including calibration-based methods (GPTQ) and calibration-free/data-free approaches (RTN, BnB, HQQ) Specifically, we consider GPTQ as a strong calibration-based 4-bit PTQ reference, and RTN (round-to-nearest) as a simple, efficient quantization baseline.

		4-bit block-wise		6-bit per-tensor	
M	**Mth.**	**QA↑**	**PPL↓**	**QA↑**	**PPL↓**
	FP	0.568	13.78	0.568	13.78
L1	GPTQ	0.518	56.88	/	/
	RTN	0.541	16.56	0.393	169.47
	BnB	**0.553**	**15.34**	/	/
	HQQ	0.544	15.75	0.404	106.02
	WGM	0.540	15.45	0.561	**14.18**
	FP	0.648	10.89	0.648	10.89
L3	GPTQ	0.633	17.09	/	/
	RTN	0.621	11.97	0.484	34.26
	BnB	**0.641**	**11.56**	/	/
	HQQ	0.638	11.67	0.373	250.72
	WGM	0.631	11.81	0.627	12.25
	FP	0.598	15.49	0.598	15.49
F1	GPTQ	**0.596**	**15.73**	/	/
	RTN	0.583	16.07	0.532	20.74
	BnB	0.593	15.97	/	/
	HQQ	0.588	15.99	0.554	18.72
	WGM	0.591	15.78	0.592	**16.26**
	FP	0.626	13.50	0.626	13.50
F3	GPTQ	**0.617**	**13.84**	/	/
	RTN	0.612	14.66	0.577	19.42
	BnB	0.608	14.20	/	/
	HQQ	0.611	14.28	0.511	20.00
	WGM	0.611	13.94	**0.627**	13.98
	FP	0.585	46.10	0.585	46.10
G1	GPTQ	0.570	59.30	/	/
	RTN	0.559	**55.60**	0.528	137.74
	BnB	**0.577**	71.05	/	/
	HQQ	0.575	59.23	0.506	161.76
	WGM	0.576	62.31	**0.583**	52.92
	FP	0.692	88.86	0.692	88.86
G4	GPTQ	0.676	105.42	/	/
	RTN	0.674	112.04	0.567	253.65
	BnB	0.682	103.32	/	/
	HQQ	0.681	**81.93**	0.579	334.60
	WGM	**0.688**	92.90	**0.694**	**94.76**

Table 1: 4-bit block-wise and 6-bit per-tensor quantization on QA and PPL. Standard errors ≤ 0.02; "/" denotes inapplicable settings. Best results are bold; our rows are shaded. L1: Llama 3.2 1B, L3: Llama 3.2 3B, F1: Falcon 3 1B, F3: Falcon 3 3B, G1: Gemma 3 1B, G4: Gemma 4B.

We further include BnB and HQQ as representative calibration-free baselines that do not require a separate calibration dataset and instead rely on weight statistics and/or a dedicated quantization objective; this makes them the closest comparisons to our approach, which also avoids calibration and auxiliary transformations. For baselines (GPTQ, RTN, BnB, HQQ), we use official implementations and/or released quantized checkpoints, following their standard configurations.

4.2 Results

We evaluate the proposed objective and solvers through a combination of proxy metrics, downstream task performance, and controlled analyses.

4.2.1 Proxy Performance and speed comparison

As shown in Table 2, WGM consistently yields the smallest weight reconstruction MSE when quantizing the first linear weight matrix of LLaMA 3.2 1B, for both per-tensor (4–6 bits) and block-wise (2–4 bits) settings. This improved reconstruction compared with baseline is obtained at the expense of higher wall-clock quantization time, with WGM being the slowest method under our CPU implementation. At the full-model scale in Table 3, WGM remains feasible using an 8-core CPU implementation across two processor tiers, whereas baseline timings rely on their standard single-GPU implementations and are therefore not directly comparable in wall-clock time.

	per-tensor			block-wise		
Mth.	**b**	**t(s)**	**MSE**	**b**	**t(s)**	**MSE**
RTN	6	0.339	170	4	1.20	69.2
RTN	5	0.252	682	3	1.11	310
RTN	4	0.245	2375	2	1.08	1586
HQQ	6	1.15	147	4	0.966	48.2
HQQ	5	1.08	590	3	0.909	216
HQQ	4	1.04	2112	2	0.902	1175
WGM	6	15.9	8.33	4	81.0	33.1
WGM	5	15.5	31.6	3	74.9	183
WGM	4	15.5	127	2	78.5	768

Table 2: Estimation of LLMs quantization performance through the first linear weight quantization MSE of meta-llama/Llama-3.2-1B following HIGGS. Per-tensor quantization

We report weight reconstruction error (Frobenius MSE) as a diagnostic proxy, motivated by recent theory linking layer-wise ℓ_2 perturbations to perplexity increase in a well-behaved regime (Malinovskii et al., 2025). However, lower weight MSE alone does not guarantee better downstream quality: prior studies show that aggressively minimizing MSE can still degrade perplexity due to overfitting (Guo et al., 2024). Accordingly, we use QA and PPL as primary evaluation metrics, and treat MSE as a *secondary* signal for numerical fidelity and fast method selection. In our experiments, MSE tends to correlate with QA/PPL across the compared settings for MSB, making it a useful preliminary proxy when full-model evaluation is costly.

4.2.2 Full QA and PPL Comparison

Our experimental results in Table 1 demonstrate that our method is competitive with widely adopted PTQ methods in block-wise scenario and outperform them in per-tensor setting.

M	GPTQ	BnB	HQQ	WGM
L1	372.9	25.6	17.7	1018.4(1054.5)
L3	780.4	52.8	34.8	3555.6(3053.9)
F1-it	428.4	24.1	18.9	1377.4(1215.2)
F3-it	772.5	35.2	35.2	2974.5(2695.5)
G1-it	399.0	19.6	13.3	777.5(703.8)
G4-it	1000.8	65.1	47.4	3633.9(3461.0)

Table 3: Quantization time (seconds). WGM is timed on 8 CPU cores (Xeon Platinum 8480+; Xeon Silver 4410Y in parentheses). Baselines are timed using their standard single-GPU implementations. Differences across the two CPU entries reflect later code optimizations; the optimized version was not rerun on the Platinum system due to access constraints.

Across six LLMs, the same objective and algorithmic pipeline is effective in both per-tensor and block-wise settings without introducing setting-specific heuristics, suggesting a granularity-agnostic formulation.

The QA score aggregates diverse reasoning and commonsense benchmarks, so consistent performance indicates robustness across heterogeneous task sensitivities rather than gains driven by a single benchmark. We also observe that GPTQ fails on Llama 3.2 1B under our evaluation setup; this behavior is unlikely to arise from implementation issues, and further analysis is provided in appendix F.

Notably, these results are obtained under a constrained PTQ pipeline, no calibration data and no auxiliary transformations, isolating the effect of the objective itself. Achieving competitive QA/PPL under these constraints highlights the practicality of the approach when calibration data are unavailable or transformations are undesirable.

Beyond averages, our method exhibits strong cross-model consistency under the block-wise setting: it achieves a top-1 rank in $1/6$, a top-2 rank in $1/6$ models for QA and $4/6$ models for PPL. Moreover, its worst-case degradation relative to the best baseline is tightly bounded, with at most $\Delta = 0.013$ in QA accuracy and $\Delta = 0.25$ in PPL (excluding Gemma, which exhibits unstable PPL even at full precision). While being the best model on all tasks under the per-tensor quantization setting.

4.2.3 Approximation algorithm performance.

To assess how close our heuristic solutions are to the optimum of the proposed objective, we additionally report oracle results for a single matrix in the same setting as Table 1 as proxy. As shown in Table 4, the exact solver achieves strictly lower

Mth.	t(s) 4b	MSE 4b	t(s) 3b	MSE 3b
DP	29800	29.96	13400	163.17
WGM	360	33.09	418	182.88

Table 4: Comparison of Estimation of LLMs quantization performance through the first linear weight quantization MSE of meta-llama/Llama-3.2-1B in block-wise setting.

MSE than WGM under identical per-tile bit-width settings (3–4 bits), with improvements of $\Delta 3.13$ and $\Delta 19.71$ for the 3-bit and 4-bit cases, respectively. However, DP requires orders-of-magnitude more computation time (hours versus seconds). We therefore use it solely as an oracle reference; it is impractical for full-model LLM quantization. See Appendix C for details.

4.2.4 Hyperparameters.

As indicated in Table 5, we observe that downstream PPL is largely stable across a broad range of λ values; we therefore set $\lambda = 0.75$ for all experiments and treat λ as a low-sensitivity hyperparameter. The rationale for choosing 0.75 instead of 0 is detailed in Appendix H; full sweeps appear in Appendix C.

λ	0.0	0.2	0.4	0.6	0.8	1.0
t(s)	340.1	385.2	377.2	370.8	364.5	364.3
Avg.	15.06	15.09	15.09	15.11	15.10	15.10

Table 5: Performance comparison for selecting λ. Utilizing Llama3.2 1B to analyze Perplexity across varying λ values with w=256 and g=256 for efficient experimentation.

t/w	$2^{11}/32$	$2^{10}/16$	$2^{9}/8$	$2^{8}/4$	$2^{7}/2$	$2^{6}/1$
MSE	72.5	71.2	68.0	63.2	53.9	33.1
t(s)	296	296	301	317	313	360

Table 6: 4-bit block-wise quantization MSE and time of weight of first linear of Llama 3.2 1B under different block t and window w size.

In per-tensor setting, the analysis of window size w in Table 7 indicates that perplexity degrades noticeably when w exceeds 64, leading us to choose $w = 64$ for an optimal balance between perplexity and quantization time. Similarly, Table 7 shows that performance saturates around a maximum group size $g = 32$, showing WGM is the best at 6-bit per-tensor quantization.

In block-wise setting, table 6 shows MSE decrease as both the window size and block size are reduced, we therefore use 64-row block size and 1 for window size in our block-wise experiments. The associated increase in runtime reflects a standard efficiency optimality trade-off from more fine-grained merging, but remains feasible in practice, so we adopt the most optimal setting.

Bit	Time	Avg.
4	429.50	5887
5	374.81	55.17
6	371.82	15.69
7	428.05	15.19
8	385.05	15.13
9	393.64	15.11
10	383.77	15.11

(a) max group

w	Time	Avg.
8	7606.56	13.99
16	3753.57	14.02
32	1900.39	14.04
64	971.81	14.00
128	571.83	14.26
256	393.64	15.11
512	296.62	19.52

(b) Window size

Table 7: Using Llama3.2 1B to analyze Perplexity performance variation for different bit length, i.e. "max group" g, with w=256 and different w with g=256

5 Conclusion

In this work, we introduced MSB PTQ, a calibration-free and transformation-free formulation for weight-only post-training quantization, together with an objective-driven grouping framework and associated solvers. By extending binary-inspired representations to multi-bit settings through a variance-based grouping objective, our approach provides a structured and flexible framework for low-bit quantization.

We demonstrate that MSB achieves strong robustness in challenging weight-only regimes, particularly in per-tensor settings, while remaining competitive with established 4-bit block-wise PTQ methods across QA and perplexity benchmarks. Our evaluation is backend-independent and isolates the effect of the quantization objective and solvers without relying on calibration or transformation techniques.

Overall, MSB highlights an objective-driven route toward calibration-free PTQ and clarifies the connection between binary-inspired representations and low-bit quantization behavior in LLMs. Future directions include: (i) developing more efficient solvers to reduce quantization time; (ii) implementing optimized CUDA kernels to enable end-to-end inference evaluation; and (iii) integrating optional calibration and transformation techniques with the MSB framework.

Limitations

Our evaluation is simulation-based: quantized weights are emulated using higher-precision arithmetic, and we do not report end-to-end inference speedups based on optimized low-bit kernels. As a result, while the quantization objective and solvers are evaluated independently of backend effects, practical deployment efficiency remains to be demonstrated.

The optimization procedure is more computationally expensive than mature GPU-based PTQ baselines, and further work is required to develop faster solvers suitable for large-scale models. In addition, our experiments are limited to models up to 4B parameters, and scaling to larger models (e.g., 7B, 13B, or beyond) remains future work.

MSB is also closely related to scalar quantization and one-dimensional clustering on weight magnitudes. Our contribution lies in the PTQ formulation, structured representation, and empirical behavior in LLM settings, rather than in the development of new clustering objectives.

Finally, by design, our method omits calibration data and transformation techniques to isolate the objective-driven formulation. Such components are orthogonal and may be integrated to further improve performance in future work.

References

Hicham Badri and Appu Shaji. 2023. Half-quadratic quantization of large machine learning models.

Yonatan Bisk, Rowan Zellers, Ronan Le bras, Jianfeng Gao, and Yejin Choi. 2020. Piqa: Reasoning about physical commonsense in natural language. *Proceedings of the AAAI Conference on Artificial Intelligence*, 34(05):7432–7439.

Christopher Clark, Kenton Lee, Ming-Wei Chang, Tom Kwiatkowski, Michael Collins, and Kristina Toutanova. 2019. BoolQ: Exploring the surprising difficulty of natural yes/no questions. In *Proceedings of the 2019 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies, Volume 1 (Long and Short Papers)*, pages 2924–2936, Minneapolis, Minnesota. Association for Computational Linguistics.

Peter Clark, Isaac Cowhey, Oren Etzioni, Tushar Khot, Ashish Sabharwal, Carissa Schoenick, and Oyvind Tafjord. 2018. Think you have solved question answering? try arc, the ai2 reasoning challenge. *Preprint*, arXiv:1803.05457.

Tim Dettmers, Mike Lewis, Younes Belkada, and Luke Zettlemoyer. 2022. Llm.int8(): 8-bit matrix multiplication for transformers at scale. *Preprint*, arXiv:2208.07339.

Tim Dettmers, Artidoro Pagnoni, Ari Holtzman, and Luke Zettlemoyer. 2023. Qlora: Efficient finetuning of quantized llms. *arXiv preprint arXiv:2305.14314*.

Abhimanyu Dubey, Abhinav Jauhri, Abhinav Pandey, Abhishek Kadian, Ahmad Al-Dahle, Aiesha Letman, Akhil Mathur, Alan Schelten, Amy Yang, Angela Fan, and 1 others. 2024. The llama 3 herd of models. *Preprint*, arXiv:2407.21783.

Falcon-LLM-Team. 2024. The falcon 3 family of open models.

Elias Frantar, Saleh Ashkboos, Torsten Hoefler, and Dan Alistarh. 2022. GPTQ: Accurate post-training compression for generative pretrained transformers. *Preprint*, arXiv:2210.17323.

Gemma Gemma-Team, Aishwarya Kamath, Johan Ferret, Shreya Pathak, Nino Vieillard, Ramona Merhej, Sarah Perrin, Tatiana Matejovicova, Alexandre Ramé, Morgane Rivière, and 1 others. 2025. Gemma 3 technical report. *arXiv preprint arXiv:2503.19786*.

Daya Guo, Dejian Yang, Haowei Zhang, Junxiao Song, Ruoyu Zhang, Runxin Xu, Qihao Zhu, Shirong Ma, Peiyi Wang, Xiao Bi, and 1 others. 2025. Deepseek-r1: Incentivizing reasoning capability in llms via reinforcement learning. *Preprint*, arXiv:2501.12948.

Yipin Guo, Yilin Lang, and Qinyuan Ren. 2024. Gptqt: Quantize large language models twice to push the efficiency. *2024 IEEE International Conference on Cybernetics and Intelligent Systems (CIS) and IEEE International Conference on Robotics, Automation and Mechatronics (RAM)*, pages 368–373.

Wei Huang, Yangdong Liu, Haotong Qin, Ying Li, Shiming Zhang, Xianglong Liu, Michele Magno, and Xiaojuan Qi. 2024. Billm: pushing the limit of post-training quantization for llms. In *Proceedings of the 41st International Conference on Machine Learning*, pages 20023–20042.

Ji Lin, Jiaming Tang, Haotian Tang, Shang Yang, Wei-Ming Chen, Wei-Chen Wang, Guangxuan Xiao, Xingyu Dang, Chuang Gan, and Song Han. 2023. Awq: Activation-aware Weight Quantization for On-Device LLM Compression and Acceleration. In *Conference on Machine Learning and Systems*.

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

S. Lloyd. 1982. Least squares quantization in pcm. *IEEE Transactions on Information Theory*, 28(2):129–137.

Vladimir Malinovskii, Andrei Panferov, Ivan Ilin, Han Guo, Peter Richtárik, and Dan Alistarh. 2025. Higgs: Pushing the Limits of Large Language Model Quantization via the Linearity Theorem. In *North American Chapter of the Association for Computational Linguistics*, pages 10857–10886.

Mitchell P. Marcus, Beatrice Santorini, and Mary Ann Marcinkiewicz. 1993. Building a large annotated corpus of English: The Penn Treebank. *Computational Linguistics*, 19(2):313–330.

J. Max. 1960. Quantizing for minimum distortion. *IRE Transactions on Information Theory*, 6(1):7–12.

James B McQueen. 1967. Some methods of classification and analysis of multivariate observations. In *Proc. of 5th Berkeley Symposium on Math. Stat. and Prob.*, pages 281–297.

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

Todor Mihaylov, Peter Clark, Tushar Khot, and Ashish Sabharwal. 2018. Can a suit of armor conduct electricity? a new dataset for open book question answering. In *Proceedings of the 2018 Conference on Empirical Methods in Natural Language Processing*, pages 2381–2391, Brussels, Belgium. Association for Computational Linguistics.

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

Mohammad Rastegari, Vicente Ordonez, Joseph Redmon, and Ali Farhadi. 2016. Xnor-net: Imagenet classification using binary convolutional neural networks. In *Computer Vision – ECCV 2016*, pages 525–542, Cham. Springer International Publishing.

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

Yuzhang Shang, Zhihang Yuan, Qiang Wu, and Zhen Dong. 2023. Pb-llm: Partially binarized large language models. *Preprint*, arXiv:2310.00034.

Wenqi Shao, Mengzhao Chen, Zhaoyang Zhang, Peng Xu, Lirui Zhao, Zhiqiang Li, Kaipeng Zhang, Peng Gao, Y. Qiao, and Ping Luo. 2023. Omniquant: Omnidirectionally Calibrated Quantization for Large Language Models. In *International Conference on Learning Representations*, volume abs/2308.13137.

Fuwen Tan, Royson Lee, Lukasz Dudziak, Shell Xu Hu, Sourav Bhattacharya, Timothy M Hospedales, Georgios Tzimiropoulos, and Brais Martínez. 2024. Mobilequant: Mobile-friendly quantization for on-device language models. In *EMNLP (Findings)*.

Rowan Zellers, Ari Holtzman, Yonatan Bisk, Ali Farhadi, and Yejin Choi. 2019. HellaSwag: Can a machine really finish your sentence? In *Proceedings of the 57th Annual Meeting of the Association for Computational Linguistics*, pages 4791–4800, Florence, Italy. Association for Computational Linguistics.

A Algorithms and Pseudocode

We present the pseudocode for the remaining three of our designed algorithms:

- Algorithms 1: Dynamic Grouping.
- Algorithms 2: Greedy Grouping.
- Algorithms WGM-L: Local optimizing Windowed Greedy Grouping.

A.1 Algorithm 1, Dynamic Grouping

This algorithm formulates the optimization problem as a dynamic programming task, systematically exploring all possible groupings to find the global optimum. Firstly, by observing that the partition with $\text{cost}(\mathcal{G}^*)$, each group corresponds to a contiguous interval in the sorted sequence, we sort all entries of $\tilde{\boldsymbol{A}}$ in ascending order while keeping track of their original indices. This is because there exists an optimal solution minimizing variance in which each group corresponds to a contiguous interval in the sorted order. We then break the problem into sub-problems. Suppose we aim to divide n elements into k groups with minimum cost. The optimal configuration must consist of the $k-1$ groups from the first i elements with minimum cost and the remaining $n-i$ as the last group

$$dp[k][n] = \min_{1 \le i < n} dp[k-1][i] + f([i\!:\!n]).$$

Where f is the optimization objective 2 for one group, dp is the dynamic programming table, with the total row of max group number and total column of max number of elements (both including 0). And $dp[k][n]$ refers to the minimum cost of partitioning n elements into k groups, $f([i\!:\!n])$ refers to the cost for the remaining elements. We prove this is correct by contradiction. Assume there is a partition of k groups for n elements with lower cost than $dp[k][n]$

$$f([0\!:\!n]) = dp[k-1][i^*] + f([i^*\!:\!n]) < dp[k][n].$$

But $dp[k][n]$ is defined as the minimum over all possible i, including i^*. This contradicts the assumption, so $dp[k][n]$ from (3) is indeed the minimum cost.

We proved the sub-problem is correct, now construct the algorithm based on Equation (3). We set the initial dp_0 as a matrix full of infinite value except the first column and row. Then We precompute prefix sums of the sorted values and their squares to evaluate the interval cost in Eq. (2) in $O(1)$ time. After filling the DP table, the optimal objective value is obtained by minimizing over the last column and the optimal group numbers is the row number of the optimal value. We store the argmin split indices in an additional table to reconstruct the optimal partition via backtracking, and compute the per-group scale parameters (e.g., absolute means) for each recovered interval to obtain the corresponding α values.

Algorithm 1 Dynamic Grouping

Sort: $\boldsymbol{A}$
sort the absolute value of input matrix $\boldsymbol{A}$
Return Sorted array of value and index in $\boldsymbol{A}$

Prefix sum: Array
prefix_sum, squared_prefix_sum $\leftarrow$ empty array
for i in Array **do**
  prefix_sum $\leftarrow$ [prefix_sum, i]
  squared_prefix_sum $\leftarrow$ [squared_prefix_sum, i^2]
end for
Return prefix_sum, squared_prefix_sum

Cost: j, k, prefix_sum, squared_prefix_sum
group_size $\leftarrow k - j$
abs_sum $\leftarrow$ prefix_sum[k] - prefix_sum[j]
abs_sum_sqr$\leftarrow$squared_prefix_sum[k]-squared_prefix_sum[j]
$\mu \leftarrow \frac{\text{abs_sum}}{\text{group_size}}$
$\sigma^2 \leftarrow \frac{\text{abs_sum_sqr}}{\text{group_size}} - \mu^2$
cost $\leftarrow$ group_sizeσ^2 + $\frac{1}{\text{group_size}}$
Return cost, μ

Dynamic Grouping: $\boldsymbol{A} \in \mathbb{R}^{m\times n}$
$dp \leftarrow dp_0, \boldsymbol{j^*}, \boldsymbol{\mu} \leftarrow \emptyset \in \mathbb{R}^{m\times n}$
$\boldsymbol{a}, \boldsymbol{i} \leftarrow$ **Sort**$(\boldsymbol{A})$
$p_1, p_2 \leftarrow$ **Prefix sum**$(\boldsymbol{a})$
for i in 1 to mn **do**
  for j in 1 to mn **do**
    $c_{min} = \text{inf}, j^* = -1, \mu^* = -1$
    for k in 0 to $j-1$ **do**
      if k in $\boldsymbol{j^*}[:i][k-1]$ **then**
        continue
      end if
      $c, \mu \leftarrow$ **Cost**(k, j, p_1, p_2)
      $c \leftarrow c + dp[i-1][k]$
      if $c < c_{min}$ **then**
        $c_{min} \leftarrow c, j^* = j, \mu^* = \mu$
      end if
    end for
    $dp[i][j] \leftarrow c_{min}, \boldsymbol{j^*}[i][j] \leftarrow j^*, \boldsymbol{\mu}[i][j] \leftarrow \mu^*$
  end for
end for
Return $dp, \boldsymbol{j^*}, \boldsymbol{\mu}$

Algorithm 2 Greedy Grouping

Greedy merging: $\boldsymbol{A} \in \mathbb{R}^{m\times n}, g$
sort absolute value of non-zero entry into an array
initial the merge cost array $[(i$ (group start), $i+1$(group end), $cost, \mu)]$ for i in range(mn - 1)

initially the ignore array
heapify the merge cost array
while len(merge cost array) $>$ g **do**
  currentMerge $\leftarrow$ heap pop
  if currentMerge in ignore array **then**
    remove currentMerge from ignore array
    continue
  end if
  push two new merges as updates
  update ignore array by two merged, invalidated by the two new merges
end while
return merge cost array

A.2 Algorithm WGM-L, Local Optimizing Windowed Greedy Merging

Algorithm 3 initializes groups with equal-size windows, producing balanced group sizes but many initial groups that can slow merging. Algorithm WGM-L instead uses equal-range binning over $[w_{\min}, w_{\max}]$, which groups numerically similar values and typically yields low within-bin variance while reducing the initial group count. This strong initialization allows WGM-L to start from a much smaller number of initial groups while remaining well aligned with the variance-based objective. Although this can create highly unbalanced bins, it substantially accelerates the greedy merging stage. To alleviate boundary artifacts induced by unbalanced bins, after the greedy merging phase, we apply a stochastic local optimization procedure that perturbs adjacent group boundaries and accepts a move only if it strictly decreases the objective. The local optimization procedure is controlled by three hyperparameters: the boundary perturbation range b, the maximum number of iterations T, and a convergence threshold ϵ. We terminate optimization when no improvement is found for a fixed number of consecutive attempts or when the improvement falls below a small threshold. In practice, the merging phase becomes faster due to the small initial group count, and the main computational cost comes from the local optimization stage

Algorithm WGM-L Local Optimizing Windowed Greedy Merging

WGM-L: $\boldsymbol{A} \in \mathbb{R}^{m \times n}, g, k, b, T, \epsilon$
sort absolute value of non-zero entry into an array
initial the merge cost array with equal range binning, bin width $\Delta = (a_{max} - a_{min})/k$ and map each magnitude $|a|$ to bin index $= \min(k-1, \lfloor (|a| - a_{min})/\Delta \rfloor)$
initially the ignore array
heapify the merge cost array
while len(merge cost array) $>$ g **do**
  currentMerge $\leftarrow$ heapop
  if currentMerge in ignore array **then**
    remove currentMerge from ignore array
    continue
  end if
  push two new neighbouring merges as updates
  update ignore array by two old merges, invalidated by the two new merges
end while
$i = 0, \varepsilon = \inf$
while $\varepsilon > \epsilon$ or $i \geq T$ **do**
  for each group **do**
    Try the move group boundary randomly within b one time. If the total loss is lower, update the group boundary and ε.
    $i++$
  end for
end while
return merge cost array

for per-tensor setting. In the block-wise setting, the local optimization stage dominates the runtime and introduces a significant computational overhead.

B Supplementary Experiments

B.1 Additional experiments setup

Since the main text shows that weight MSE is proportional with downstream quantization quality, we use MSE as a proxy in controlled ablations. We evaluate the proposed algorithms on synthetic matrices sampled from $\mathcal{N}(0, 1)$ and run all experiments on a single CPU. These standardized test cases allow us to compare quantization loss, runtime, and hyperparameter sensitivity using MSE as a proxy.

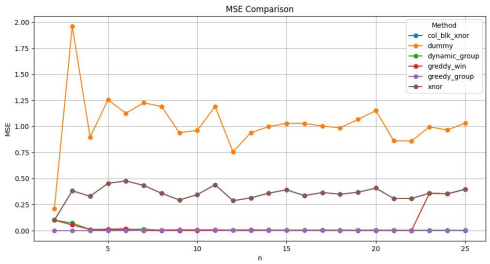

Figure 2: Algorithm comparison for small matrix, quantization loss against matrix sized $\mathbb{R}^{n \times n}$.

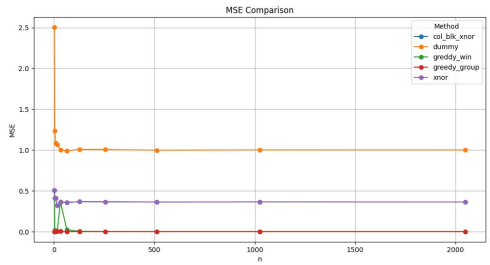

Figure 3: Algorithm comparison for large matrix, quantization loss against matrix sized $\mathbb{R}^{n \times n}$.

B.2 Comparison for quantization loss

We evaluate the three core variants (Algorithms 1, 2, and 3) by proxy MSE on synthetic square matrices. Figures 2 and 3 compare against XNOR, BLOCKED-XNOR, and an all-zero dummy baseline. As expected, the all-zero dummy yields the largest error, while XNOR and BLOCKED-XNOR produce moderate MSE. In contrast, our methods consistently reduce MSE to near zero on these controlled instances.

In Figure 2, Algorithm 3 eventually matches XNOR. This is an implementation artifact of the dynamic window schedule: when the window size grows to n, the matrix dimension, windowed merging degenerates to standard XNOR. For sufficiently large inputs, the code increases the window automatically, causing this convergence behavior.

In Figure 3, we omit Algorithm 1 due to prohibitive runtime at larger sizes. We evaluate square matrices with side length $n \in \{2, 4, \ldots, 2048\}$. As n grows, the loss curves become smoother analogs of the small-scale behavior, suggesting that the qualitative trends generalize. Moreover, Algorithms 2 and 3 closely track the optimum provided by Algorithm 1, indicating that their approximation gap is negligible in terms of MSE on these instances. The relative ordering of errors typically follows our expectation (DG $\leq$ GG $\leq$ WGM), but

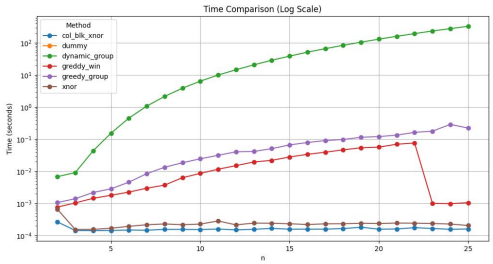

Figure 4: Algorithm comparison for small matrix, time used for quantization on CPU against matrix sized $\mathbb{R}^{n\times n}$.

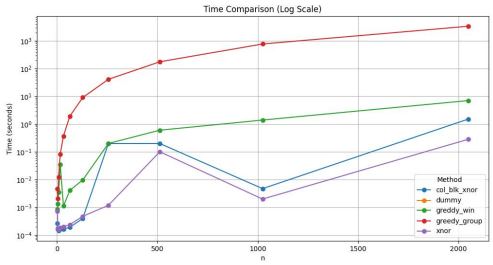

Figure 5: Algorithm comparison for large matrix, time used for quantization on CPU against matrix sized $\mathbb{R}^{n\times n}$.

the absolute differences remain small.

We omit Algorithm 4 (WGM-L) from this proxy ablation because the goal here is to benchmark heuristic grouping strategies against the dynamic-programming oracle (Algorithm 1). Algorithm 3 already achieves MSE nearly indistinguishable from the oracle on these controlled instances, establishing that its approximation gap is negligible in the proxy regime. In contrast, WGM-L is a local refinement applied on top of Algorithm 3 and it is feasible to run to completion on LLMs, we therefore evaluate WGM-L directly against Algorithm 3 using full PPL/QA rather than the MSE proxy.

B.3 Comparison for quantization speed

Following the setup in the previous section, we replace the MSE proxy with wall-clock quantization time and compare runtime on both small and large square matrices. In Figure 4, XNOR and BLOCKED-XNOR are the fastest baselines, as they rely on lightweight mean/sign computations. Among our methods, Algorithms 2 and 3 are substantially faster than the oracle Algorithm 1, consistent with the fact that Algorithm 1 trades increased computational complexity for optimality.

The abrupt runtime change observed for Algorithm 3 is due to its dynamic window schedule: once the window size grows to the matrix dimension n, the method degenerates to standard XNOR, leading to a sudden speed-up.

As the matrix size increases, runtime grows rapidly and Algorithm 1 becomes impractical. Figure 5 reports the same comparison on larger matrices. Algorithm 2 can require on the order of 10^3 seconds at the largest size, whereas Algorithm 3 completes within ~ 10 seconds, i.e., over two orders of magnitude faster. XNOR and BLOCKED-XNOR remain the fastest, but their advantage narrows as the matrix size grows. Taken together with its near-oracle loss in the proxy study, Algorithm 3 offers the best accuracy–efficiency trade-off and is therefore used as the default variant in subsequent experiments.

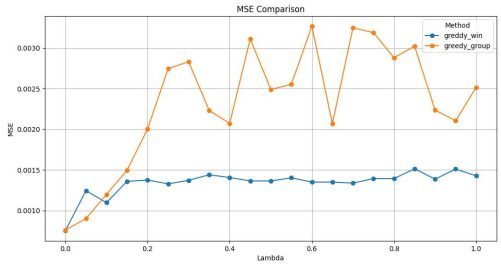

Figure 6: Loss against λ for greedy algorithm and windowed version.

B.4 Comparison for loss against λ for greedy algorithm and windowed version

Figure 6 reports the MSE against λ on a synthetic matrix in $\mathbb{R}^{512\times 512}$, chosen as a compromise between computational cost and performance. We evaluate only Algorithm 2 and Algorithm 3, since Algorithm 1 is not computationally feasible at this scale and Algorithm WGM-L is a modified Algorithm 3. Varying λ has limited practical effect for Algorithm 2 and Algorithm 3, where the targeted number of groups is specified externally rather than optimized through λ. In particular, both methods attain their best MSE at $\lambda = 0$, and Algorithm 3 and WGM-L shows negligible variation and no general trends across the entire range of λ respectively, mirroring the weak dependence observed in downstream evaluations in the main text. These results suggest that, outside Algorithm 1, λ does not serve as an effective control knob for trading performance and efficiency.

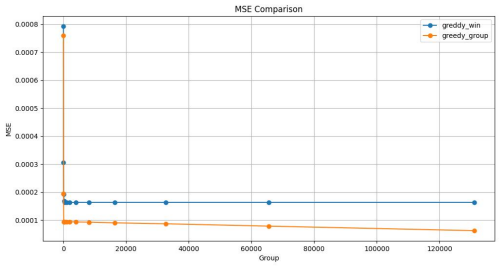

Figure 7: Max groups against MSE.

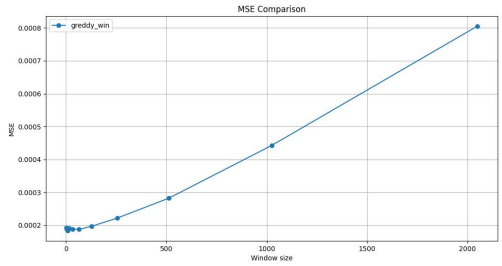

Figure 9: Window (initial group) size against loss.

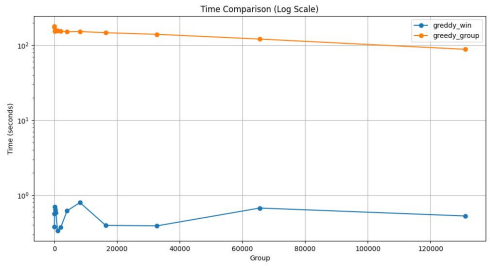

Figure 8: Max groups against quantization speed.

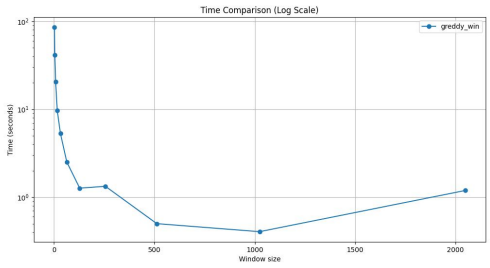

Figure 10: Window (initial group) size against quantization speed.

B.5 Comparison for max groups against MSE and speed

Since we need to choose the max group manually, we evaluate its impact on quantization error and speed using matrix $\in \mathbb{R}^{512\times512}$ to discover the best operating bit length in Figure 7 and 8. As shown in the figures, the MSE for Algorithm 2 and 3 improves and then stabilizes, plateaus around 32 groups, while quantization time decreases slowly as the max group increases.

B.6 Comparison for window (initial group) size against quantization loss and speed.

Lastly, the impact of different window sizes in Algorithm 3 on MSE and quantization speed was evaluated in the matrix simulation experiments, as shown in Figure 9 and 10. It was observed that increasing the window size led to higher MSE and reduced quantization time. When the window size was below 64, the MSE remained near its minimum, while the speed improvement flattened out between 64 and 1024. Therefore, a window size of 64 is the best balance point.

C Expanded Tables

For completeness, we provide full versions of several condensed tables from the main text. Specifically, Table 7 is expanded into Tables 8 and 9; Table 6 is expanded into Tables 11 and 12; and Table 5 is expanded into Table 10. The qualitative trends are consistent with the condensed results in the main text. The time in parentheses is parallel overhead.

g	b	Time	WK2	PTB	C4	Avg.
8	4	429.50(177.39)	7677	5165	4819	5887
16	5	374.81(128.15)	12.61	24.05	18.51	55.17
32	6	371.82 (127.25)	10.94	20.21	15.92	15.69
64	7	428.05(174.66)	10.65	19.45	15.45	15.19
128	8	385.05(135.06)	10.62	19.37	15.39	15.13
256	9	393.64(145.25)	10.62	19.36	15.37	15.11
512	10	383.77(145.77)	10.62	19.35	15.36	15.11

Table 8: Performance comparison for selecting "max group" g. Utilizing Llama3.2 1B to analyze Perplexity across varying g values with w=256 for efficient experimentation.

D Supplementary PPL and QA

For completeness, we provide full versions of the condensed results in Table 1. Tables 13 and 14 report per-task QA scores and per-dataset perplexities for the block-wise 4-bit setting, respectively, whose averages correspond to the 4-bit columns in Table 1. Similarly, Tables 21 and 22 provide the corresponding breakdown for the per-tensor 6-bit setting. All qualitative trends are consistent with the condensed table in the main text.

w	Time	WK2	PTB	C4	Avg.
8	7606.56 (139.23)	9.90	17.83	14.22	13.99
16	3753.57 (147.63)	9.90	17.91	14.24	14.02
32	1900.39 (134.34)	9.90	17.98	14.24	14.04
64	971.81 (133.88)	9.92	17.83	14.24	14.00
128	571.83 (130.67)	10.11	18.08	14.59	14.26
256	393.64 (145.25)	10.62	19.36	15.37	15.11
512	296.62 (130.72)	12.42	27.90	18.25	19.52

Table 9: Performance comparison for selecting window size w, using Llama3.2 1B to analyze perplexity across varying w values, with g = 256 for efficient experimentation. b=9

λ	Time	WK2	PTB	C4	Avg.
0.0	340.10 (111.23)	10.60	19.24	15.34	15.06
0.1	391.23 (132.98)	10.61	19.29	15.37	15.09
0.2	385.21 (133.20)	10.61	19.29	15.36	15.09
0.3	394.20 (134.01)	10.61	19.27	15.36	15.08
0.4	377.17 (130.04)	10.61	19.29	15.38	15.09
0.5	386.47 (129.55)	10.61	19.28	15.37	15.09
0.6	370.80 (126.68)	10.62	19.32	15.39	15.11
0.7	378.87 (127.96)	10.62	19.35	15.38	15.11
0.8	364.53 (130.19)	10.62	19.34	15.36	15.10
0.9	374.72 (133.59)	10.62	19.34	15.39	15.12
1.0	364.32 (129.50)	10.61	19.32	15.37	15.10

Table 10: Performance comparison for selecting λ. Utilizing Llama3.2 1B to analyze Perplexity across varying λ values with w=256 and g=256 for efficient experimentation.

We also provide full QA/PPL tables in the same settings for instruction-tuned (-it) models for block-wise 4-bit and for per-tensor 5-bit/4-bit. The instruction-tuned block-wise 4-bit results follow the same trends as the base models. Per-tensor 5-bit slightly degrades WGM, while HQQ and RTN collapse; per-tensor 4-bit causes substantial failure across all methods in our evaluation pipeline. Collectively, these results indicate that the comparative trends and failure modes are largely stable across a broad range of settings.

In all main experiments, WGM-L uses $T = 12$ local optimization steps and initial group count $k = 256$.

We evaluate WGM-L only in the per-tensor setting, where its coarse-to-fine design is most effective for large-scale grouping. In contrast, for block-wise quantization, the base WGM already operates with minimal window size and smaller problem instances, making the additional coarse

w \ t	2048	1024	512	256	128	64
64	82.43	107.7	177.5	394.0	2290	/
32	72.50	81.51	105.8	176.1	391.3	2272
16	70.33	71.23	78.61	103.4	173.0	385.6
8	69.43	68.41	67.97	74.85	98.11	167.2
4	69.15	67.13	64.34	63.20	67.70	88.76
2	68.98	66.29	62.64	57.88	53.88	53.84
1	68.69	66.04	61.21	54.46	44.93	33.09

Table 11: 4-bit per tile quantization MSE of weight of first linear of Llama 3.2 1B under different block t and window w size

w \ t	2048	1024	512	256	128	64
64	292.4	289.7	288.2	289.7	306.3	/
32	295.5	291.2	295.5	297.2	288.9	324.1
16	296.8	296.0	297.5	301.7	308.0	325.3
8	305.6	291.6	301.2	288.8	312.6	289.8
4	304.3	320.5	293.2	317.2	322.8	290.0
2	361.4	356.0	350.8	349.2	312.6	336.0
1	394.9	410.0	385.1	404.1	406.3	360.5

Table 12: 4-bit per tile quantization time of weight of first linear of Llama 3.2 1B under different block t and window w size

initialization unnecessary.

Empirically, in Table 21 WGM-L achieves improved performance on a subset of models (3/6 in both QA and PPL) while remaining comparable in others, indicating that local boundary refinement can provide modest gains in per-tensor quantization quality. Algorithm WGM-L summarizes the procedure.

In terms of complexity, the greedy merging stage of WGM-L operates in $O(k \log k)$ time, where k is the number of initial groups. This is followed by a local optimization stage with complexity $O(Tg)$, where g denotes the final number of groups and T the maximum number of refinement iterations.

Abbreviations. We use the following abbreviations for QA benchmarks: AC (ARC-C), AE (ARC-E), BQ (BoolQ), HS (HellaSwag), OQ (OPQA), PQ (PIQA), WG (WinoGrande).

M	Mth.	b	AC ↑	AE ↑	BQ ↑	HS ↑	OQ ↑	PQ ↑	WG ↑	Avg. ↑
L1	FP	16	0.364	0.605	0.639	0.636	0.374	0.744	0.612	0.568
	GPTQ	4	0.334	0.561	0.578	0.455	**0.372**	0.723	0.600	0.518
	RTN	4	0.347	0.558	0.626	0.600	0.356	0.727	0.571	0.541
	BnB	4	**0.350**	**0.587**	**0.635**	**0.610**	0.348	0.737	0.602	**0.553**
	HQQ	4	0.348	0.553	0.596	0.605	0.358	**0.741**	**0.607**	0.544
	WGM	4	0.343	0.570	0.625	0.607	0.330	0.725	0.583	0.540
L3	FP	16	0.458	0.716	0.728	0.736	0.432	0.775	0.693	0.648
	GPTQ	4	0.444	0.694	0.708	0.722	0.426	0.763	0.677	0.633
	RTN	4	0.431	0.662	0.699	0.720	0.400	0.771	0.663	0.621
	BnB	4	0.447	**0.713**	0.707	**0.728**	**0.428**	**0.777**	0.684	**0.641**
	HQQ	4	**0.452**	0.694	**0.734**	0.722	0.412	0.769	0.685	0.638
	WGM	4	0.434	0.683	0.726	0.714	0.406	0.761	**0.694**	0.631
F1	FP	16	0.421	0.662	0.719	0.618	0.406	0.745	0.617	0.598
	GPTQ	4	**0.422**	0.662	**0.720**	0.607	**0.408**	0.744	0.605	**0.596**
	RTN	4	0.398	0.642	0.696	0.607	0.402	0.745	0.590	0.583
	BnB	4	0.421	**0.671**	0.718	0.606	0.386	0.744	0.601	0.593
	HQQ	4	0.410	0.645	0.697	**0.612**	0.396	**0.747**	**0.607**	0.588
	WGM	4	0.415	0.662	0.700	0.611	**0.408**	0.746	0.598	0.591
F3	FP	16	0.472	0.727	0.735	0.652	0.394	0.751	0.650	0.626
	GPTQ	4	0.451	0.709	0.740	**0.640**	0.382	**0.757**	**0.640**	**0.617**
	RTN	4	0.444	0.703	**0.764**	0.631	0.374	0.736	0.632	0.612
	BnB	4	0.448	0.709	0.710	0.635	0.372	0.748	0.632	0.608
	HQQ	4	0.451	0.696	0.725	0.632	**0.388**	0.746	0.635	0.611
	WGM	4	**0.455**	**0.714**	0.703	0.635	0.384	0.751	0.638	0.611
G1	FP	16	0.381	0.722	0.663	0.622	0.374	0.749	0.584	0.585
	GPTQ	4	0.356	0.703	0.640	0.610	0.358	0.743	0.584	0.570
	RTN	4	0.354	0.700	0.603	0.597	0.350	0.730	0.580	0.559
	BnB	4	0.367	0.694	**0.656**	0.605	**0.374**	0.742	**0.602**	**0.577**
	HQQ	4	0.369	**0.705**	0.646	0.604	0.364	**0.745**	0.590	0.575
	WGM	4	**0.370**	0.703	0.647	**0.617**	**0.374**	0.742	0.581	0.576
G4	FP	16	0.547	0.817	0.789	0.758	0.434	0.803	0.695	0.692
	GPTQ	4	0.506	0.793	0.778	0.744	0.426	0.799	0.685	0.676
	RTN	4	0.507	0.798	0.760	0.743	0.424	**0.800**	0.684	0.674
	BnB	4	0.526	0.798	0.776	0.747	**0.452**	0.791	0.683	0.682
	HQQ	4	**0.529**	0.808	0.770	0.738	0.444	0.794	0.682	0.681
	WGM	4	**0.529**	**0.817**	**0.786**	**0.754**	0.434	0.799	**0.698**	**0.688**

Table 13: QA performance of pretrained LLMs under 4-bit block-wise quantization across seven benchmarks.

M	Mth.	b	WK2 ↓	PTB ↓	C4 ↓	Avg. ↓
L1	FP	16	9.75	17.59	14.01	13.78
	GPTQ	4	40.87	64.06	65.72	56.88
	RTN	4	11.58	20.99	17.10	16.56
	BnB	4	**10.78**	**19.43**	**15.82**	**15.34**
	HQQ	4	11.07	19.90	16.29	15.75
	WGM	4	10.89	19.47	15.98	15.45
L3	FP	16	7.81	13.53	11.33	10.89
	GPTQ	4	12.23	23.60	15.43	17.09
	RTN	4	8.57	14.66	12.68	11.97
	BnB	4	**8.28**	**14.24**	**12.15**	**11.56**
	HQQ	4	8.32	14.35	12.32	11.67
	WGM	4	8.43	14.54	12.49	11.81
F1	FP	16	9.15	19.67	17.65	15.49
	GPTQ	4	**9.27**	**20.04**	**17.89**	**15.73**
	RTN	4	9.46	20.46	18.30	16.07
	BnB	4	9.45	20.36	18.11	15.97
	HQQ	4	9.42	20.41	18.14	15.99
	WGM	4	9.31	20.08	17.96	15.78
F3	FP	16	8.02	16.49	15.99	13.50
	GPTQ	4	**8.21**	**16.96**	**16.35**	**13.84**
	RTN	4	8.63	18.22	17.15	14.66
	BnB	4	8.39	17.47	16.73	14.20
	HQQ	4	8.45	17.59	16.81	14.28
	WGM	4	8.26	17.10	16.47	13.94
G1	FP	16	14.17	104.12	20.00	46.10
	GPTQ	4	15.46	140.91	**21.52**	59.30
	RTN	4	16.37	**127.22**	23.21	**55.60**
	BnB	4	17.22	172.16	23.77	71.05
	HQQ	4	16.85	137.64	23.20	59.23
	WGM	4	**15.39**	149.58	21.95	62.31
G4	FP	16	10.77	239.02	16.78	88.86
	GPTQ	4	11.66	286.68	17.94	105.42
	RTN	4	12.27	305.39	18.45	112.04
	BnB	4	11.48	281.24	**17.23**	103.32
	HQQ	4	11.28	**217.25**	17.25	**81.93**
	WGM	4	**11.19**	249.96	17.55	92.90

Table 14: PPL performance of pretrained LLMs under 4-bit block-wise quantization across three datasets.

M	Mth.	b	WK2 ↓	PTB ↓	C4 ↓	Avg. ↓
L1-it	FP	16	13.16	25.69	21.31	20.05
	GPTQ	4	16.86	30.03	26.35	24.41
	RTN	4	16.22	31.91	26.59	24.91
	BnB	4	**14.67**	**28.80**	**23.37**	**22.28**
	HQQ	4	15.05	29.19	24.18	22.81
	WGM	4	15.25	29.39	23.90	22.85
L3-it	FP	16	11.04	20.42	16.49	15.98
	GPTQ	4	13.07	25.48	17.93	18.83
	RTN	4	11.84	21.50	17.70	17.01
	BnB	4	**11.67**	**20.55**	**17.16**	**16.46**
	HQQ	4	11.70	21.85	17.42	16.99
	WGM	4	12.02	24.55	17.47	18.01
F1-it	FP	16	10.51	22.29	19.47	17.42
	GPTQ	4	27.11	63.73	62.39	51.08
	RTN	4	10.72	23.07	20.08	17.96
	BnB	4	10.73	22.67	19.86	17.75
	HQQ	4	10.75	22.74	19.87	17.79
	WGM	4	**10.42**	**22.51**	**19.62**	**17.51**
F3-it	FP	16	10.11	20.19	18.27	16.19
	GPTQ	4	15.57	42.65	40.17	32.80
	RTN	4	10.51	21.21	19.18	16.97
	BnB	4	10.57	21.20	19.16	16.98
	HQQ	4	**9.96**	**20.90**	**18.86**	**16.57**
	WGM	4	11.01	21.66	20.31	17.66
G1-it	FP	16	27.67	212.14	33.21	91.01
	GPTQ	4	**31.00**	**251.43**	38.00	**106.81**
	RTN	4	37.07	362.66	43.29	147.67
	BnB	4	33.90	311.52	39.89	128.44
	HQQ	4	34.77	310.36	40.58	128.57
	WGM	4	31.71	266.08	**37.33**	111.71
G4-it	FP	16	17.36	644.68	23.38	228.47
	GPTQ	4	19.03	818.43	25.48	287.65
	RTN	4	18.48	**558.52**	25.64	**200.88**
	BnB	4	19.82	811.48	25.92	285.74
	HQQ	4	**18.25**	642.86	**24.53**	228.55
	WGM	4	18.34	736.24	**24.53**	259.71

Table 15: PPL performance of instruction finetuned LLMs under 4-bit block-wise quantization across three datasets.

M	Mth.	b	AC ↑	AE ↑	BQ ↑	HS ↑	OQ ↑	PQ ↑	WG ↑	Avg. ↑
L1-it	FP	16	0.379	0.634	0.692	0.608	0.344	0.745	0.603	0.572
	GPTQ	4	**0.363**	0.573	0.674	0.577	0.344	0.706	0.576	0.545
	RTN	4	0.343	0.609	0.673	0.565	0.332	0.718	0.581	0.546
	BnB	4	**0.363**	**0.616**	**0.684**	0.585	0.340	**0.730**	0.577	**0.556**
	HQQ	4	0.357	0.609	0.664	0.585	**0.346**	0.728	**0.591**	0.554
	WGM	4	**0.363**	0.607	0.668	**0.587**	0.336	0.720	0.577	0.551
L3-it	FP	16	0.460	0.678	0.784	0.704	0.360	0.756	0.675	0.631
	GPTQ	4	0.433	0.658	0.753	0.694	0.360	0.745	0.657	0.614
	RTN	4	0.431	0.639	**0.779**	0.696	**0.372**	**0.753**	**0.680**	**0.621**
	BnB	4	**0.447**	0.641	0.770	0.699	0.368	0.746	0.668	0.620
	HQQ	4	0.433	0.657	0.762	**0.701**	0.370	0.749	0.662	0.619
	WGM	4	0.437	**0.681**	0.734	0.686	**0.372**	0.732	0.655	0.614
F1-it	FP	16	0.453	0.680	0.732	0.631	0.404	0.750	0.601	0.607
	GPTQ	4	0.442	0.678	**0.731**	0.626	0.396	0.747	0.594	0.602
	RTN	4	0.436	0.675	0.712	0.626	0.404	0.737	0.599	0.598
	BnB	4	0.455	0.670	0.728	0.625	0.408	0.742	**0.609**	0.605
	HQQ	4	0.452	0.643	0.710	0.624	**0.410**	0.740	0.586	0.595
	WGM	4	**0.459**	**0.683**	0.728	**0.627**	0.402	**0.751**	0.606	**0.608**
F3-it	FP	16	0.544	0.749	0.788	0.701	0.422	0.759	0.649	0.659
	GPTQ	4	0.506	0.716	0.775	**0.680**	0.406	**0.754**	**0.648**	0.641
	RTN	4	0.495	0.734	0.783	0.671	0.410	0.738	0.625	0.637
	BnB	4	0.506	0.732	**0.789**	0.679	0.410	0.752	0.626	0.642
	HQQ	4	**0.521**	**0.751**	0.787	0.676	0.414	0.748	0.637	**0.648**
	WGM	4	0.480	0.735	0.720	0.638	**0.416**	0.749	0.595	0.619
G1-it	FP	16	0.385	0.637	0.759	0.578	0.384	0.724	0.589	0.579
	GPTQ	4	0.363	0.606	**0.738**	0.536	**0.374**	0.709	**0.599**	0.561
	RTN	4	0.338	0.564	0.712	0.536	0.360	0.714	0.575	0.543
	BnB	4	0.356	0.609	0.703	**0.562**	0.366	**0.725**	0.579	0.557
	HQQ	4	0.353	0.583	0.717	0.551	0.370	0.720	0.587	0.555
	WGM	4	**0.394**	**0.611**	0.735	**0.562**	0.368	0.724	0.584	**0.568**
G4-it	FP	16	0.571	0.779	0.839	0.744	0.468	0.774	0.695	0.696
	GPTQ	4	0.539	0.765	0.827	0.727	0.450	0.771	0.666	0.678
	RTN	4	0.532	**0.779**	0.826	0.733	0.448	0.773	**0.680**	0.681
	BnB	4	0.529	0.772	**0.840**	0.731	0.464	**0.779**	**0.680**	0.685
	HQQ	4	0.542	0.764	0.829	0.732	0.460	0.768	0.669	0.681
	WGM	4	**0.561**	0.754	0.831	**0.738**	**0.468**	0.772	0.679	**0.686**

Table 16: QA performance of instruction finetuned LLMs under 4-bit block-wise quantization across seven benchmarks.

M	Mth.	b	AC ↑	AE ↑	BQ ↑	HS ↑	OQ ↑	PQ ↑	WG ↑	Avg. ↑
L1	FP	16	0.364	0.605	0.639	0.636	0.374	0.744	0.612	0.568
	RTN	4	0.257	0.254	**0.600**	0.263	**0.298**	0.517	0.500	**0.384**
	HQQ	4	**0.263**	0.256	0.540	**0.265**	0.286	0.511	0.490	0.373
	WGM	4	0.240	**0.281**	0.381	0.262	0.276	0.511	0.497	0.350
	WGM-L	4	0.237	0.279	0.403	0.264	0.246	**0.520**	**0.502**	0.350
L3	FP	16	0.458	0.716	0.728	0.736	0.432	0.775	0.693	0.648
	RTN	4	**0.266**	0.251	**0.598**	0.263	**0.290**	0.502	**0.526**	**0.385**
	HQQ	4	0.253	0.244	0.564	0.265	0.278	**0.529**	0.492	0.375
	WGM	4	0.257	**0.269**	0.381	0.268	0.280	0.519	0.515	0.356
	WGM-L	4	0.251	0.268	0.378	**0.269**	0.274	0.508	0.503	0.350
F1	FP	16	0.421	0.662	0.719	0.618	0.406	0.745	0.617	0.598
	RTN	4	0.271	0.286	0.519	0.266	0.288	0.508	0.500	0.377
	HQQ	4	0.253	0.263	0.459	0.262	0.282	0.512	0.491	0.360
	WGM	4	0.279	**0.467**	0.500	**0.406**	0.298	**0.648**	**0.526**	**0.446**
	WGM-L	4	**0.281**	0.429	**0.549**	0.379	**0.306**	0.628	0.504	0.439
F3	FP	16	0.472	0.727	0.735	0.652	0.394	0.751	0.650	0.626
	RTN	4	0.239	0.265	0.386	0.264	0.314	0.494	0.515	0.354
	HQQ	4	0.244	0.255	0.452	0.259	0.292	0.523	0.532	0.365
	WGM	4	**0.315**	**0.534**	**0.618**	**0.483**	**0.328**	**0.683**	**0.536**	**0.499**
	WGM-L	4	0.271	0.435	0.508	0.384	0.298	0.642	0.510	0.435
G1	FP	16	0.381	0.722	0.663	0.622	0.374	0.749	0.584	0.585
	RTN	4	0.255	0.256	0.457	0.259	0.298	0.503	0.504	0.362
	HQQ	4	0.252	0.270	0.442	0.266	0.292	0.526	0.485	0.362
	WGM	4	**0.366**	0.633	0.529	0.609	0.370	0.720	0.563	0.541
	WGM-L	4	0.361	**0.659**	**0.603**	**0.617**	**0.380**	**0.725**	**0.579**	**0.561**
G4	FP	16	0.547	0.817	0.789	0.758	0.434	0.803	0.695	0.692
	RTN	4	0.265	0.253	0.499	0.262	0.284	0.494	0.506	0.366
	HQQ	4	0.261	0.263	0.549	0.267	0.290	0.506	0.497	0.376
	WGM	4	**0.413**	0.668	0.575	**0.631**	0.390	**0.749**	**0.620**	**0.578**
	WGM-L	4	0.408	**0.678**	**0.596**	0.622	**0.394**	0.738	0.598	0.576

Table 17: QA performance of pretrained LLMs under 4-bit per-tensor quantization across seven benchmarks.

M	Mth.	b	WK2 ↓	PTB ↓	C4 ↓	Avg. ↓
L1	FP	16	9.75	17.59	14.01	13.78
	RTN	4	2057063.25	1895821.88	2303753.25	2085546.12
	HQQ	4	3482935.25	2573738.25	3059556.75	3038743.42
	WGM	4	28731.11	20510.02	12215.58	20485.57
	WGM-L	4	15098.34	12288.76	7531.40	11639.50
L3	FP	16	7.81	13.53	11.33	10.89
	RTN	4	257957.31	96724.30	479669.94	278117.18
	HQQ	4	1227376.50	1219055.50	1298070.25	1248167.42
	WGM	4	10806.46	12205.56	6340.89	9784.30
	WGM-L	4	9696.50	10285.10	4389.21	8123.60
F1	FP	16	9.15	19.67	17.65	15.49
	RTN	4	2917153.50	4835918.00	6318141.00	4690404.17
	HQQ	4	940907.81	2635117.00	1690569.75	1755531.52
	WGM	4	77.70	185.41	158.94	140.68
	WGM-L	4	120.04	266.01	263.32	216.46
F3	FP	16	8.02	16.49	15.99	13.50
	RTN	4	11405618.00	17293052.00	19399576.00	16032748.67
	HQQ	4	1336339.75	4439809.00	4443197.50	3406448.75
	WGM	4	21.05	43.04	39.46	34.52
	WGM-L	4	73.75	93.26	136.10	101.04
G1	FP	16	14.17	104.12	20.00	46.10
	RTN	4	21185708.00	31281078.00	12719446.00	21728744.00
	HQQ	4	9682463.00	24361728.00	14554462.00	16199551.00
	WGM	4	49.78	797.77	67.11	304.89
	WGM-L	4	55.29	1004.90	64.69	374.96
G4	FP	16	10.77	239.02	16.78	88.86
	RTN	4	43750133760.00	254194106368.00	173915586560.00	157286608896.00
	HQQ	4	11806313472.00	1974162176.00	15918100480.00	9899525376.00
	WGM	4	34.70	1890.78	55.13	660.21
	WGM-L	4	36.44	2768.71	55.18	953.44

Table 18: PPL performance of pretrained LLMs under 4-bit per-tensor quantization across three datasets.

M	Mth.	b	AC ↑	AE ↑	BQ ↑	HS ↑	OQ ↑	PQ ↑	WG ↑	Avg. ↑
L1	FP	16	0.364	0.605	0.639	0.636	0.374	0.744	0.612	0.568
	RTN	5	0.258	0.256	0.385	0.260	0.272	0.516	0.498	0.349
	HQQ	5	0.272	0.264	0.381	0.257	0.284	0.493	0.511	0.352
	WGM	5	**0.353**	**0.587**	0.569	**0.617**	**0.350**	**0.724**	0.578	**0.540**
	WGM-L	5	0.346	0.524	**0.606**	0.596	0.346	0.715	**0.579**	0.530
L3	FP	16	0.458	0.716	0.728	0.736	0.432	0.775	0.693	0.648
	RTN	5	**0.265**	0.267	0.378	0.264	**0.282**	0.481	0.508	0.349
	HQQ	5	0.264	0.262	0.423	0.263	**0.282**	0.508	0.492	0.356
	WGM	5	0.212	0.317	0.394	0.289	0.244	0.551	0.511	0.360
	WGM-L	5	0.219	**0.389**	**0.466**	**0.310**	0.256	**0.610**	**0.525**	**0.396**
F1	FP	16	0.421	0.662	0.719	0.618	0.406	0.745	0.617	0.598
	RTN	5	0.293	0.465	0.480	0.415	0.296	0.636	0.534	0.446
	HQQ	5	0.304	0.557	0.635	0.462	0.290	0.677	0.541	0.495
	WGM	5	**0.399**	**0.646**	**0.699**	**0.600**	**0.400**	**0.742**	**0.594**	**0.583**
	WGM-L	5	0.397	0.643	0.683	0.562	0.376	0.726	0.579	0.567
F3	FP	16	0.472	0.727	0.735	0.652	0.394	0.751	0.650	0.626
	RTN	5	0.272	0.402	0.469	0.381	0.284	0.588	0.518	0.416
	HQQ	5	0.246	0.317	0.541	0.292	0.272	0.547	0.491	0.386
	WGM	5	**0.442**	0.683	**0.747**	0.618	0.382	0.741	0.623	0.605
	WGM-L	5	0.437	**0.689**	0.737	**0.619**	**0.396**	**0.748**	**0.634**	**0.608**
G1	FP	16	0.381	0.722	0.663	0.622	0.374	0.749	0.584	0.585
	RTN	5	0.222	0.292	0.389	0.271	0.270	0.533	0.482	0.351
	HQQ	5	0.231	0.330	0.490	0.294	0.270	0.545	0.500	0.380
	WGM	5	0.382	0.710	0.652	0.628	**0.364**	0.744	**0.581**	0.580
	WGM-L	5	**0.383**	**0.711**	**0.661**	**0.632**	0.360	**0.748**	0.579	**0.582**
G4	FP	16	0.547	0.817	0.789	0.758	0.434	0.803	0.695	0.692
	RTN	5	0.255	0.283	0.437	0.277	0.260	0.529	0.509	0.364
	HQQ	5	0.244	0.303	0.474	0.288	0.254	0.521	0.518	0.372
	WGM	5	**0.529**	**0.807**	**0.747**	0.738	**0.436**	**0.798**	**0.676**	**0.676**
	WGM-L	5	0.501	0.790	**0.747**	**0.747**	0.418	0.797	**0.677**	0.668

Table 19: QA performance of pretrained LLMs under 5-bit per-tensor quantization across seven benchmarks.

M	Mth.	b	WK2 ↓	PTB ↓	C4 ↓	Avg. ↓
L1	FP	16	9.75	17.59	14.01	13.78
	RTN	5	160604.80	80296.23	112862.91	117921.32
	HQQ	5	152015.38	89273.23	93242.31	111510.31
	WGM	5	11.32	20.28	16.47	16.02
	WGM-L	5	13.21	26.06	19.43	19.57
L3	FP	16	7.81	13.53	11.33	10.89
	RTN	5	196842.44	115724.54	87181.27	133249.41
	HQQ	5	23792.54	31359.14	22844.35	25998.68
	WGM	5	2325.18	2728.98	2814.28	2622.81
	WGM-L	5	446.43	548.06	764.34	586.28
F1	FP	16	9.15	19.67	17.65	15.49
	RTN	5	55.00	131.81	96.32	94.38
	HQQ	5	24.87	52.11	46.64	41.21
	WGM	5	10.37	23.75	20.08	18.07
	WGM-L	5	17.32	46.43	31.88	31.88
F3	FP	16	8.02	16.49	15.99	13.50
	RTN	5	82.58	189.23	148.03	139.95
	HQQ	5	387.51	705.79	457.26	516.85
	WGM	5	9.53	20.92	18.92	16.46
	WGM-L	5	9.49	20.60	18.81	16.30
G1	FP	16	14.17	104.12	20.00	46.10
	RTN	5	2226.50	20169.60	1670.33	8022.14
	HQQ	5	1868.31	12820.58	1592.09	5426.99
	WGM	5	17.20	236.83	23.94	92.66
	WGM-L	5	15.71	162.77	22.39	66.96
G4	FP	16	10.77	239.02	16.78	88.86
	RTN	5	42114252.00	99411056.00	4541906.50	48689071.50
	HQQ	5	20682.83	958119.62	9247.24	329349.90
	WGM	5	13.21	553.78	20.95	195.98
	WGM-L	5	12.65	531.05	20.10	187.93

Table 20: PPL performance of pretrained LLMs under 5-bit per-tensor quantization across three datasets.

M	Mth.	b	AC ↑	AE ↑	BQ ↑	HS ↑	OQ ↑	PQ ↑	WG ↑	Avg. ↑
L1	FP	16	0.364	0.605	0.639	0.636	0.374	0.744	0.612	0.568
	RTN	6	0.228	0.331	0.551	0.327	0.224	0.563	0.527	0.393
	HQQ	6	0.240	0.379	0.439	0.398	0.258	0.591	0.520	0.404
	WGM	6	**0.369**	**0.596**	0.636	**0.636**	**0.366**	0.732	0.590	0.561
	WGM-L	6	0.359	0.592	**0.640**	0.632	0.362	**0.747**	**0.600**	**0.562**
L3	FP	16	0.458	0.716	0.728	0.736	0.432	0.775	0.693	0.648
	RTN	6	0.298	0.445	0.585	0.544	0.312	0.633	0.571	0.484
	HQQ	6	0.224	0.282	0.545	0.279	0.248	0.524	0.510	0.373
	WGM	6	0.444	0.695	0.667	0.720	0.408	0.770	0.685	0.627
	WGM-L	6	**0.463**	**0.701**	**0.698**	**0.733**	**0.434**	**0.779**	**0.699**	**0.644**
F1	FP	16	0.421	0.662	0.719	0.618	0.406	0.745	0.617	0.598
	RTN	6	0.363	0.593	0.570	0.558	0.348	0.708	0.585	0.532
	HQQ	6	0.375	0.629	0.582	0.583	0.384	0.736	0.591	0.554
	WGM	6	0.421	0.663	**0.700**	**0.614**	0.402	0.745	0.596	0.592
	WGM-L	6	**0.426**	**0.670**	0.697	0.613	**0.406**	**0.748**	**0.604**	**0.595**
F3	FP	16	0.472	0.727	0.735	0.652	0.394	0.751	0.650	0.626
	RTN	6	0.401	0.649	0.713	0.583	0.374	0.727	0.595	0.577
	HQQ	6	0.393	0.633	0.668	0.540	0.342	0.703	0.576	0.551
	WGM	6	0.469	**0.715**	**0.761**	**0.654**	**0.398**	**0.755**	0.637	**0.627**
	WGM-L	6	**0.473**	0.714	0.735	0.653	0.382	0.752	**0.648**	0.622
G1	FP	16	0.381	0.722	0.663	0.622	0.374	0.749	0.584	0.585
	RTN	6	0.333	0.629	0.565	0.536	0.342	0.713	0.575	0.528
	HQQ	6	0.321	0.623	0.448	0.549	0.338	0.713	0.551	0.506
	WGM	6	**0.386**	**0.722**	**0.664**	**0.623**	0.358	0.747	**0.583**	**0.583**
	WGM-L	6	0.373	0.714	0.663	0.621	**0.368**	**0.748**	0.568	0.579
G4	FP	16	0.547	0.817	0.789	0.758	0.434	0.803	0.695	0.692
	RTN	6	0.375	0.641	0.562	0.626	0.382	0.746	0.639	0.567
	HQQ	6	0.403	0.693	0.536	0.647	0.382	0.754	0.635	0.579
	WGM	6	0.545	0.817	**0.790**	**0.764**	0.436	**0.808**	**0.696**	**0.694**
	WGM-L	6	**0.554**	**0.821**	0.776	0.760	**0.444**	0.804	0.687	0.692

Table 21: QA performance of pretrained LLMs under 6-bit per-tensor quantization across seven benchmarks.

M	Mth.	b	WK2 ↓	PTB ↓	C4 ↓	Avg. ↓
L1	FP	16	9.75	17.59	14.01	13.78
	RTN	6	132.9	190.7	184.8	169.5
	HQQ	6	82.94	133.0	102.2	106.0
	WGM	6	**10.04**	**18.04**	**14.45**	**14.18**
	WGM-L	6	10.09	18.16	14.57	14.27
L3	FP	16	7.81	13.53	11.33	10.89
	RTN	6	22.26	48.53	32.01	34.26
	HQQ	6	148.6	261.9	341.6	250.7
	WGM	6	8.53	15.24	12.97	12.25
	WGM-L	6	**7.99**	**13.82**	**11.64**	**11.15**
F1	FP	16	9.15	19.67	17.65	15.49
	RTN	6	12.28	26.58	23.37	20.74
	HQQ	6	10.92	24.13	21.12	18.72
	WGM	6	**9.55**	**20.87**	**18.35**	**16.26**
	WGM-L	6	9.82	21.93	18.97	16.91
F3	FP	16	8.02	16.49	15.99	13.50
	RTN	6	11.38	24.64	22.24	19.42
	HQQ	6	11.83	24.98	23.20	20.00
	WGM	6	8.27	17.25	16.44	13.98
	WGM-L	6	**8.13**	**16.84**	**16.18**	**13.72**
G1	FP	16	14.17	104.1	20.00	46.10
	RTN	6	29.04	345.1	39.10	137.7
	HQQ	6	31.11	414.5	39.70	161.8
	WGM	6	**14.37**	124.2	**20.21**	52.92
	WGM-L	6	14.49	**111.6**	20.69	**48.94**
G4	FP	16	10.77	239.0	16.78	88.86
	RTN	6	24.46	703.5	32.97	253.7
	HQQ	6	22.70	947.4	33.72	334.6
	WGM	6	**10.77**	**256.6**	**16.90**	**94.76**
	WGM-L	6	10.96	275.9	17.03	101.3

Table 22: PPL performance of pretrained LLMs under 6-bit per-tensor quantization across three datasets.

E Double Quantization

We further evaluate a double-quantization variant that quantizes the quantization parameters, scaler, in addition to the weights recursively once more using the same WGM algorithm. Tables 23 and 24 compare QA and PPL between single and double quantization. The effect is stable across all tested models: double quantization consistently leads to a small degradation (lower QA and higher PPL) relative to single quantization. Since the observed changes are uniform and do not improve end-task quality, we focus on single quantization in the main paper and include double quantization here as a supplementary analysis.

F GPTQ Behavior on Llama 3.2 1B Pretrained

So we evaluated using GPTQ for Llama-3.2 1B pretrained check point found on hugging face, namely shuyuej/Llama-3.2-1B-GPTQ (GPTQ A) and saul95/Llama-3.2-1B-GPTQ (GPTQ B). Then we also used known correct implementation, GPTQModel[1], to quantize on Llama 3.2 1B (GPTQ C). Across these variants, GPTQ exhibited substantially degraded performance under our evaluation protocol, suggesting that the observed behavior is unlikely to be an artifact of a single repository, checkpoint, or invocation.

The exact cause remains unclear at the time of submission. One possible explanation is sensitivity to the calibration procedure. GPTQ relies on small calibration sets, and misalignment between calibration data and the downstream evaluation distribution can lead to overfitting or instability in certain models or settings. In contrast, calibration-free PTQ approaches do not condition on such data and often display more uniform behavior across architectures and datasets, as observed in prior reports (Malinovskii et al., 2025; Guo et al., 2024; Lin et al., 2023). We emphasize that this remains a hypothesis; a systematic study varying calibration sample size, selection strategy, and layer-wise objectives would be required to establish causality.

[1] https://github.com/ModelCloud/GPTQModel

M	Mth.	b	AC ↑	AE ↑	BQ ↑	HS ↑	OQ ↑	PQ ↑	WG ↑	Avg. ↑
L1	WGM	4	0.343	0.570	0.625	0.607	0.330	0.725	0.583	0.540
	WGM-dq	4	0.335	0.567	0.597	0.600	0.332	0.719	0.588	0.534
L1-it	WGM	4	0.363	0.607	0.668	0.587	0.336	0.720	0.577	0.551
	WGM-dq	4	0.347	0.596	0.651	0.572	0.328	0.705	0.563	0.537
L3	WGM	4	0.434	0.683	0.726	0.714	0.406	0.761	0.694	0.631
	WGM-dq	4	0.381	0.615	0.516	0.625	0.380	0.741	0.652	0.559
L3-it	WGM	4	0.437	0.681	0.734	0.686	0.372	0.732	0.655	0.614
	WGM-dq	4	0.361	0.593	0.559	0.583	0.336	0.695	0.596	0.532
F1	WGM	4	0.415	0.662	0.700	0.611	0.408	0.746	0.598	0.591
	WGM-dq	4	0.411	0.666	0.709	0.608	0.410	0.745	0.597	0.592
F1-it	WGM	4	0.459	0.683	0.728	0.627	0.402	0.751	0.606	0.608
	WGM-dq	4	0.457	0.684	0.730	0.625	0.398	0.744	0.609	0.607
F3	WGM	4	0.455	0.714	0.703	0.635	0.384	0.751	0.638	0.611
	WGM-dq	4	0.452	0.716	0.701	0.630	0.388	0.751	0.646	0.612
F3-it	WGM	4	0.480	0.735	0.720	0.638	0.416	0.749	0.595	0.619
	WGM-dq	4	0.478	0.723	0.710	0.637	0.414	0.743	0.595	0.614
G1	WGM	4	0.370	0.703	0.647	0.617	0.374	0.742	0.581	0.576
	WGM-dq	4	0.369	0.697	0.621	0.614	0.366	0.747	0.579	0.571
G1-it	WGM	4	0.394	0.611	0.735	0.562	0.368	0.724	0.584	0.568
	WGM-dq	4	0.384	0.629	0.735	0.558	0.362	0.719	0.568	0.565
G4	WGM	4	0.529	0.817	0.786	0.754	0.434	0.799	0.698	0.688
	WGM-dq	4	0.532	0.821	0.793	0.756	0.434	0.795	0.697	0.690
G4-it	WGM	4	0.561	0.754	0.831	0.738	0.468	0.772	0.679	0.686
	WGM-dq	4	0.552	0.743	0.829	0.737	0.462	0.768	0.676	0.681

Table 23: QA performance of double quantization of pretrained LLMs using WGM in 4-bit block-wise quantization. Double quantization are on blocks of quantization parameters of size 2048, bit=6. so on average each parameter cost 6 + 32*16/2048=6.25 bits and the whole matrix cost 4 + 8*6.25/64=4.78125 bits.

M	Mth.	b	WK2 ↓	PTB ↓	C4 ↓	Avg. ↓
L1	WGM	4	10.89	19.47	15.98	15.45
	WGM dq	4	11.17	19.92	16.39	15.83
L1-it	WGM	4	15.25	29.39	23.90	22.85
	WGM dq	4	16.71	32.06	25.90	24.89
L3	WGM	4	8.43	14.54	12.49	11.81
	WGM dq	4	16.12	33.39	26.62	25.37
L3-it	WGM	4	12.02	24.55	17.47	18.01
	WGM dq	4	20.88	73.80	40.00	44.89
F1	WGM	4	9.31	20.08	17.96	15.78
	WGM-dq	4	9.37	20.18	18.05	15.87
F1-it	WGM	4	10.42	22.51	19.62	17.51
	WGM-dq	4	10.51	22.59	19.69	17.60
F3	WGM	4	8.26	17.10	16.47	13.94
	WGM-dq	4	8.36	17.32	16.60	14.09
F3-it	WGM	4	11.01	21.66	20.31	17.66
	WGM-dq	4	11.47	22.33	20.85	18.21
G1	WGM	4	246.69	1111.53	330.01	562.75
	WGM-dq	4	253.96	1225.12	332.91	604.00
G1-it	WGM	4	538.23	1477.78	553.44	856.49
	WGM-dq	4	534.46	1451.77	533.80	840.01
G4	WGM	4	58.66	369.94	91.59	173.40
	WGM-dq	4	63.87	387.97	99.62	183.82
G4-it	WGM	4	86.06	822.51	112.66	340.41
	WGM-dq	4	85.01	849.22	112.73	348.97

Table 24: PPL performance of double quantization of pretrained LLMs using WGM in 4-bit block-wise quantization. With the same setting.

M	Mth.	b	AC ↑	AE ↑	BQ ↑	HS ↑	OQ ↑	PQ ↑	WG ↑	Avg. ↑
L1	FP	16	0.364	0.605	0.639	0.636	0.374	0.744	0.612	0.568
	GPTQ A	4	0.317	0.553	0.502	0.357	0.354	0.694	0.583	0.480
	GPTQ B	4	0.334	0.561	0.578	0.455	0.372	0.723	0.600	0.518
	GPTQ C	4	0.253	0.479	0.509	0.358	0.298	0.658	0.552	0.444

Table 25: QA of GPTQ A,B and C

M	Mth.	b	WK2 ↓	PTB ↓	C4 ↓	Avg. ↓
L1	FP	16	9.75	17.59	14.01	13.78
	GPTQ A	4	195.34	213.91	302.27	237.18
	GPTQ B	4	40.87	64.06	65.72	56.88
	GPTQ C	4	113.16	131.91	176.33	140.47

Table 26: PPL of GPTQ A,B and C

G Detailed Derivation of Proposed Methods

Given the objective

$$g^*, \{\boldsymbol{A}_i^*\}_{i=1}^{g^*} = \underset{\substack{\{\boldsymbol{A}_i\}_{i=1}^{g} \in \mathbb{A} \\ \{\boldsymbol{B}_i\}_{i=1}^{g} \in \mathbb{B} \\ 1 \leq g \leq |\boldsymbol{A}|}}{\arg\min} \sum_{i=1}^{g} ||\boldsymbol{A}_i - \alpha_i \boldsymbol{B}_i||_2^2 + \frac{1}{|\boldsymbol{B}_i|}$$

Since $|\boldsymbol{A}_i| = |\boldsymbol{B}_i|$, the optimization objective can be rewritten to

$$g^*, \{\boldsymbol{A}_i^*\}_{i=1}^{g^*} = \underset{\substack{\{\boldsymbol{A}_i\}_{i=1}^{g} \in \mathbb{A} \\ \{\boldsymbol{B}_i\}_{i=1}^{g} \in \mathbb{B} \\ 1 \leq g \leq |\boldsymbol{A}|}}{\arg\min} \sum_{i=1}^{g} ||\boldsymbol{A}_i - \alpha_i \boldsymbol{B}_i||_2^2 + \frac{1}{|\boldsymbol{A}_i|}$$

As shown in 1,

$$\underset{\alpha^*, \boldsymbol{B}^*}{\arg\min} ||\boldsymbol{A} - \alpha \boldsymbol{B}||_2^2, \quad \alpha^* = \frac{||\boldsymbol{A}||_1}{|\boldsymbol{A}|}, \ \boldsymbol{B}^* = \text{sign}(\boldsymbol{A})$$

By substituting the optimal values, we can rewrite the squared 2-norm.

$$||\boldsymbol{A} - \alpha^* \boldsymbol{B}^*||_2^2 = \sum^{i} \sum^{j} (A_{i,j} - \alpha^* \cdot \text{sign}(A_{i,j}))^2$$

$$
\begin{aligned}
&= \sum^{i}\sum^{j}(|A_{i,j}| - \alpha^*)^2 \\
&= \sum^{i}\sum^{j}|A_{i,j}|^2 - 2\alpha^*\sum^{i}\sum^{j}|A_{i,j}| + \sum^{i}\sum^{j}\alpha^{*2} \\
&= ||\boldsymbol{A}||_2^2 - 2\alpha^*||\boldsymbol{A}||_1 + |\boldsymbol{A}|\alpha^{*2} \\
&= ||\boldsymbol{A}||_2^2 - 2\frac{||\boldsymbol{A}||_1}{|\boldsymbol{A}|}||\boldsymbol{A}||_1 + |\boldsymbol{A}|\left(\frac{||\boldsymbol{A}||_1}{|\boldsymbol{A}|}\right)^2 \\
&= ||\boldsymbol{A}||_2^2 - 2\frac{||\boldsymbol{A}||_1^2}{|\boldsymbol{A}|} + \frac{||\boldsymbol{A}||_1^2}{|\boldsymbol{A}|} \\
&= ||\boldsymbol{A}||_2^2 - \frac{||\boldsymbol{A}||_1^2}{|\boldsymbol{A}|}
\end{aligned}
$$

Consider the variance of absolute value of $\boldsymbol{A}_i$, let $\tilde{\boldsymbol{A}}_i$ be $\boldsymbol{A}_i$ with element-wise absolute value, such as $|\tilde{\boldsymbol{A}}_i| = |\boldsymbol{A}_i|$ and $\tilde{A}_{i_{j,k}} = |A_{i_{j,k}}|$.

$$
\begin{aligned}
Var(\tilde{\boldsymbol{A}}_i) &= E[\tilde{\boldsymbol{A}}_i^{\,2}] - E[\tilde{\boldsymbol{A}}_i]^2 \\
&= \frac{1}{|\boldsymbol{A}_i|}\sum^{j}\sum^{k}|A_{i_{j,k}}|^2 - \left(\frac{1}{|\boldsymbol{A}_i|}\sum^{j}\sum^{k}|A_{i_{j,k}}|\right)^2 \\
&= \frac{||\boldsymbol{A}_i||_2^2}{|\boldsymbol{A}_i|} - \left(\frac{||\boldsymbol{A}_i||_1}{|\boldsymbol{A}_i|}\right)^2
\end{aligned}
$$

$$
|\boldsymbol{A}_i|Var(\tilde{\boldsymbol{A}}_i) = ||\boldsymbol{A}_i||_2^2 - \frac{||\boldsymbol{A}_i||_1^2}{|\boldsymbol{A}_i|}
$$

Together, we can conclude

$$
||\boldsymbol{A}_i - \alpha_i^*\boldsymbol{B}_i^*||_2^2 = |\boldsymbol{A}_i|Var(\tilde{\boldsymbol{A}}_i)
$$

Finally, we rewrite the initial optimization objective as

$$
g^*, \{\boldsymbol{A}_i^*\}_{i=1}^{g^*} = \underset{\substack{\{\boldsymbol{A}_i\}_{i=1}^{g}\in\mathbb{A} \\ 1\le g\le|\boldsymbol{A}|}}{\arg\min}\sum_{i=1}^{g}|\boldsymbol{A}_i|\text{Var}(\tilde{\boldsymbol{A}}_i) + \frac{1}{|\boldsymbol{A}_i|}
$$

H Determining the Boundary of λ

We further analysis the boundary of λ, the objective in this section is

$$
\min_{g\in\mathbb{N}}\sum_{i=1}^{g}\left(\tfrac{|\boldsymbol{A}_i|}{|\boldsymbol{A}|}Var(\tilde{\boldsymbol{A}}_i) + \tfrac{\lambda}{|\boldsymbol{A}_i|}\right) \tag{4}
$$

Firstly, we determine the lower bound of λ. Considering the case of group size 1 and number of groups g is the number of non-zero elements n. The cost is

$$
\sum^{g} 0 + \lambda\frac{1}{1} = n\lambda \tag{5}
$$

The squared 2 norm is 0 since in the case of group size one $\boldsymbol{A}_i = \alpha_i\boldsymbol{B}_i$.

Now consider the case of the number of groups = $g-1$, so we have $g-2$ groups of size 1 and 1 group of size 2. The cost is

$$
\begin{aligned}
&\sum^{g-2}\left(0+\lambda\frac{1}{1}\right)+\frac{1}{n}(|a_1|-\alpha^*)^2+\frac{1}{n}(|a_2|-\alpha^*)^2+\frac{\lambda}{2}\\
=&\sum^{n-2}\left(0+\lambda\frac{1}{1}\right)+\frac{1}{n}(|a_1|-\frac{|a_1|+|a_2|}{2})^2\\
&+\frac{1}{n}(|a_2|-\frac{|a_1|+|a_2|}{2})^2+\frac{\lambda}{2}\\
=&(n-2)\lambda+\frac{1}{n}(\frac{|a_1|-|a_2|}{2})^2+\frac{1}{n}(\frac{|a_2|-|a_1|}{2})^2+\frac{\lambda}{2}\\
=&(n-2)\lambda+\frac{1}{n}(\frac{|a_1|-|a_2|}{2})^2+\frac{1}{n}(\frac{|a_1|-|a_2|}{2})^2+\frac{\lambda}{2}\\
=&(n-2)\lambda+\frac{(|a_1|-|a_2|)^2}{2n}+\frac{\lambda}{2} \qquad (6)
\end{aligned}
$$

Since we are minimizing the objective of equation 4, when (5) $<$ (6) we have n groups, else at most n-1 groups. Because we are using quantization for higher efficiency, a smaller number of groups, each with a larger size is desired. Therefore, we must avoid the n group case, i.e.(5) $>$ (6).

$$
\begin{aligned}
n\lambda &> (n-2)\lambda+\frac{(|a_1|-|a_2|)^2}{2n}+\frac{\lambda}{2}\\
2\lambda &> \frac{(|a_1|-|a_2|)^2}{2n}+\frac{\lambda}{2}\\
\frac{3\lambda}{2} &> \frac{(|a_1|-|a_2|)^2}{2n}\\
\lambda &> \frac{(|a_1|-|a_2|)^2}{3n}\\
\lambda &> \min_{1\le i<n}\frac{(|a_i|-|a_{i+1}|)^2}{3n}
\end{aligned}
$$

Since we also want to avoid the n-1 group case, and any group number that leads to a small group size, in general. So a good λ will not be in the close neighbourhood of the actual lower bound, thus we can estimate the lower bound roughly as

$$
\lambda > \frac{(|a_1|-|a_2|)^2}{3n} > \min_{1\le i<n}\frac{(|a_i|-|a_{i+1}|)^2}{3n}
$$

We can just use the first two elements in the sorted array for fast computation and estimation of the lower bound of λ, so we have a lower bound

$$
\lambda > \frac{(|a_1|-|a_2|)^2}{3n} \qquad (7)
$$

Next, we deduce the upper bound for λ with a similar method. Considering the case of one group, the cost is

$$
\begin{aligned}
&\frac{1}{|\boldsymbol{A}|}|\boldsymbol{A}|Var(\tilde{\boldsymbol{A}})+\frac{\lambda}{|\boldsymbol{A}|}\\
=&\frac{1}{n}nVar(\tilde{\boldsymbol{A}})+\frac{\lambda}{n}
\end{aligned}
$$

$$=Var(\tilde{\boldsymbol{A}}) + \frac{\lambda}{n} \tag{8}$$

Considering two group of matrix $\boldsymbol{A}_1, \boldsymbol{A}_2$ split at k^{th} sorted non-zero element of $\boldsymbol{A}$, the cost is

$$\begin{aligned}
&\frac{1}{|\boldsymbol{A}|}|\boldsymbol{A}_1|Var(\tilde{\boldsymbol{A}}_1) + \frac{\lambda}{|\boldsymbol{A}_1|} + \frac{1}{|\boldsymbol{A}|}|\boldsymbol{A}_2|Var(\tilde{\boldsymbol{A}}_2) + \frac{\lambda}{|\boldsymbol{A}_2|} \\
=&\frac{1}{n}\left(kVar(\tilde{\boldsymbol{A}}_1) + (n-k)Var(\tilde{\boldsymbol{A}}_2)\right) + \frac{\lambda}{k} + \frac{\lambda}{n-k}
\end{aligned} \tag{9}$$

If we want to have at least 2 group, (9) $<$ (8),

$$\begin{aligned}
&\frac{1}{n}\left(k\,\text{Var}(\tilde{\boldsymbol{A}}_1) + (n-k)\,\text{Var}(\tilde{\boldsymbol{A}}_2)\right) + \frac{\lambda}{k} + \frac{\lambda}{n-k} < \text{Var}(\tilde{\boldsymbol{A}}) + \frac{\lambda}{n} \\
&k\,\text{Var}(\tilde{\boldsymbol{A}}_1) + (n-k)\,\text{Var}(\tilde{\boldsymbol{A}}_2) + \frac{n\lambda}{k} + \frac{n\lambda}{n-k} < n\,\text{Var}(\tilde{\boldsymbol{A}}) + \lambda \\
&\frac{n\lambda}{k} + \frac{n\lambda}{n-k} - \lambda < n\,\text{Var}(\tilde{\boldsymbol{A}}) - \left(k\,\text{Var}(\tilde{\boldsymbol{A}}_1) + (n-k)\,\text{Var}(\tilde{\boldsymbol{A}}_2)\right) \\
&\lambda\left(\frac{n}{k} + \frac{n}{n-k} - 1\right) < n\,\text{Var}(\tilde{\boldsymbol{A}}) - \left(k\,\text{Var}(\tilde{\boldsymbol{A}}_1) + (n-k)\,\text{Var}(\tilde{\boldsymbol{A}}_2)\right) \\
&\lambda < \frac{n\,\text{Var}(\tilde{\boldsymbol{A}}) - \left(k\,\text{Var}(\tilde{\boldsymbol{A}}_1) + (n-k)\,\text{Var}(\tilde{\boldsymbol{A}}_2)\right)}{\frac{n}{k} + \frac{n}{n-k} - 1}
\end{aligned}$$

Notice we can simplify the above expression by rewriting $nVar(\tilde{\boldsymbol{A}})$

$$nVar(\tilde{\boldsymbol{A}}) = \sum_{x\in\tilde{\boldsymbol{A}}}(x-\mu)^2 = \sum_{x_1\in\tilde{\boldsymbol{A}}_1}(x_1-\mu)^2 + \sum_{x_2\in\tilde{\boldsymbol{A}}_2}(x_2-\mu)^2$$

Where μ is the average of elements of $\tilde{\boldsymbol{A}}$. Notice

$$\begin{aligned}
&\sum_{x_1\in\tilde{\boldsymbol{A}}_1}(x_1-\mu)^2 = \sum_{x_1\in\tilde{\boldsymbol{A}}_1}((x_1-\mu_1)+(\mu_1-\mu))^2 \\
&= \sum_{x_1\in\tilde{\boldsymbol{A}}_1}(x_1-\mu_1)^2 + 2(x_1-\mu_1)(\mu_1-\mu) + (\mu_1-\mu)^2 \\
&= \left(\sum_{x_1\in\tilde{\boldsymbol{A}}_1}\left((x_1-\mu_1)^2 + (\mu_1-\mu)^2\right)\right) + \\
&\quad \left(2(\mu_1-\mu)\sum_{x_1\in\tilde{\boldsymbol{A}}_1}(x_1-\mu_1)\right) \\
&= \left(\sum_{x_1\in\tilde{\boldsymbol{A}}_1}\left((x_1-\mu_1)^2 + (\mu_1-\mu)^2\right)\right) + \\
&\quad \left(2(\mu_1-\mu)\left(\left(\sum_{x_1\in\tilde{\boldsymbol{A}}_1}x_1\right) - |\boldsymbol{A}_1|\mu_1\right)\right) \\
&= \left(\sum_{x_1\in\tilde{\boldsymbol{A}}_1}\left((x_1-\mu_1)^2 + (\mu_1-\mu)^2\right)\right) + (2(\mu_1-\mu)(0))
\end{aligned}$$

$$
\begin{aligned}
&= \left(\sum_{x_1 \in \tilde{\boldsymbol{A}}_1} \left((x_1 - \mu_1)^2 + (\mu_1 - \mu)^2 \right) \right) \\
&= \sum_{x_1 \in \tilde{\boldsymbol{A}}_1} (x_1 - \mu_1)^2 + k(\mu_1 - \mu)^2
\end{aligned}
$$

Similarly

$$
\begin{aligned}
\sum_{x_2 \in \tilde{\boldsymbol{A}}_2} (x_2 - \mu)^2 &= \sum_{x_2 \in \tilde{\boldsymbol{A}}_2} \left((x_2 - \mu_2)^2 + (\mu_2 - \mu)^2 \right) \\
&= \sum_{x_2 \in \tilde{\boldsymbol{A}}_2} (x_2 - \mu_2)^2 + (n-k)(\mu_2 - \mu)^2
\end{aligned}
$$

Substitute back

$$
\begin{aligned}
& nVar(\tilde{\boldsymbol{A}}) \\
=& \sum_{x_1 \in \tilde{\boldsymbol{A}}_1} (x_1 - \mu_1)^2 + k(\mu_1 - \mu)^2 + \\
& \sum_{x_2 \in \tilde{\boldsymbol{A}}_2} (x_2 - \mu_2)^2 + (n-k)(\mu_2 - \mu)^2 \\
=& kVar(\tilde{\boldsymbol{A}}_1) + (n-k)Var(\tilde{\boldsymbol{A}}_2) + k(\mu_1 - \mu)^2 + (n-k)(\mu_2 - \mu)^2
\end{aligned}
$$

Substituting this into the upper bound, we get

$$
\begin{aligned}
\lambda &< \frac{nVar(\tilde{\boldsymbol{A}}) - \left(kVar(\tilde{\boldsymbol{A}}_1) + (n-k)Var(\tilde{\boldsymbol{A}}_2) \right)}{\frac{n}{k} + \frac{n}{n-k} - 1} \\
&= \frac{k(\mu_1 - \mu)^2 + (n-k)(\mu_2 - \mu)^2}{\frac{n}{k} + \frac{n}{n-k} - 1} \\
&< \max_{1 \le k < n} \frac{k(\mu_1 - \mu)^2 + (n-k)(\mu_2 - \mu)^2}{\frac{n}{k} + \frac{n}{n-k} - 1}
\end{aligned}
$$

We can further simplify the numerator.

$$
\begin{aligned}
& k(\mu_1 - \mu)^2 + (n-k)(\mu_2 - \mu)^2 \\
=& k(\mu_1 - \frac{k\mu_1 + (n-k)\mu_2}{n})^2 + (n-k)(\mu_2 - \frac{k\mu_1 + (n-k)\mu_2}{n})^2 \\
=& k(\frac{n\mu_1 - k\mu_1 - (n-k)\mu_2}{n})^2 + (n-k)(\frac{n\mu_2 - k\mu_1 - (n-k)\mu_2}{n})^2 \\
=& k(\frac{(n-k)\mu_1 - (n-k)\mu_2}{n})^2 + (n-k)(\frac{-k\mu_1 + k\mu_2}{n})^2 \\
=& k(\frac{(n-k)(\mu_1 - \mu_2)}{n})^2 + (n-k)(\frac{-k(\mu_1 - \mu_2)}{n})^2 \\
=& \frac{k(n-k)^2(\mu_1 - \mu_2)^2 + (n-k)k^2(\mu_1 - \mu_2)^2}{n^2} \\
=& \frac{kn - k^2 + k^2}{n^2}(n-k)(\mu_1 - \mu_2)^2
\end{aligned}
$$

$$=\frac{k}{n}(n-k)(\mu_1-\mu_2)^2$$

So the upper bound is

$$\lambda < \max_{1\le k<n} \frac{\frac{k}{n}(n-k)(\mu_1-\mu_2)^2}{\frac{n}{k}+\frac{n}{n-k}-1}$$

To balance efficiency and accuracy, we avoid approaching the precise upper bound, as it maximizes efficiency at the cost of minimal accuracy. Instead, a faster approximation is employed to estimate the upper bound. Using $k=\frac{n}{2}$

$$\begin{aligned}
\lambda &< \frac{\frac{\frac{n}{2}}{n}(n-\frac{n}{2})(\mu_1-\mu_2)^2}{\frac{n}{\frac{n}{2}}+\frac{n}{n-\frac{n}{2}}-1} < \max_{1\le k<n} \frac{\frac{k}{n}(n-k)(\mu_1-\mu_2)^2}{\frac{n}{k}+\frac{n}{n-k}-1} \\
\lambda &< \frac{\frac{1}{2}(\frac{n}{2})(\mu_1-\mu_2)^2}{2+\frac{n}{\frac{n}{2}}-1} \\
&= \frac{\frac{n}{4}(\mu_1-\mu_2)^2}{3} \\
&= \frac{n(\mu_1-\mu_2)^2}{12}
\end{aligned}$$

Finally, we have

$$\lambda_{min} \approx \frac{(|a_1|-|a_2|)^2}{3n} < \lambda < \frac{n(\mu_1-\mu_2)^2}{12} \approx \lambda_{max} \tag{10}$$

And we can control λ through $\tilde{\lambda} \in [0,1]$ by

$$\begin{aligned}
\lambda &= \Lambda(\tilde{\lambda}) \\
&= \lambda_{min} + \tilde{\lambda}(\lambda_{max}-\lambda_{min}) \\
&= \frac{(|a_1|-|a_2|)^2}{3n} + \tilde{\lambda}(\frac{n(\mu_1-\mu_2)^2}{12} - \frac{(|a_1|-|a_2|)^2}{3n})
\end{aligned}$$

For the optimal λ, the hypothesis is that a suitable value of λ should lie within the latter half of the range, but not too close to the upper bound, which prioritizes efficiency over accuracy. A precise choice of λ leads to a specific optimal max group, which can be identified via binary search over the result max group of Algorithm 1. Since this process is too computationally intensive, we hypothesize that $\tilde{\lambda}^* \approx 0.75$ provides a reasonable approximation. Empirical results on Algorithm 2 and 3, however, indicate that λ has negligible impact on quantization performance of PPL. The same holds for $\tilde{\lambda}$, and the corresponding table is omitted, as it is virtually identical to Table 5. This observation is theoretically supported: λ is originally introduced to determine the optimal number of groups in Algorithm 1, whereas in other algorithms, the number of groups is treated as a user-defined hyperparameter, rendering it inapplicable. For consistency of notation and to preserve the theoretical formulation of Algorithm 1, we keep the λ term throughout.