ADAPIVOT: AN ADAPTIVE CORE-TAIL REPRESENTATION FOR LOW-BIT LLM QUANTIZATION

Anonymous authors
Paper under double-blind review

ABSTRACT

Post-training quantization (PTQ) of Large Language Models (LLMs) often suffers substantial accuracy degradation because fixed numerical formats poorly accommodate the pronounced outliers in weights and activations. Through extensive profiling across diverse models, we observe a distributional structure: most values concentrate in a compact core, while the outliers, which account for only a small fraction of the values, form a sparse tail, typically separated from the core by a sharp magnitude transition and a nearly unoccupied interval. Building on this observation, we introduce AdaPivot, an adaptive low-bit numerical format that partitions the representable range into distinct core and tail regions. AdaPivot independently adapts the representation granularity of the two regions, together with their boundary and dynamic ranges, to match the underlying tensor distribution within a compact parametric form. We further formulate format fitting as a joint discrete–continuous reconstruction problem and develop a tailored solver that efficiently co-optimizes discrete level selection and continuous format parameters. An integer-constrained variant further reduces metadata overhead while preserving the adaptive structure. Experiments on various models from 7B to 70B parameters show that AdaPivot consistently outperforms representative PTQ baselines. Under W4A4 quantization, it improves average zero-shot accuracy by up to 7.5% over the state-of-the-art baseline, while W4A8 performance closely approaches the FP16 baselines. Our code is available at https://anonymous.4open.science/r/AdaPivot-A.

1 INTRODUCTION

The rapid development and broad adoption of Large Language Models (LLMs) have enabled strong performance across a wide range of applications (Brown et al., 2020; Achiam et al., 2023; Touvron et al., 2023; Vaswani et al., 2017), but their growing parameter counts and compute- and memory-intensive inference make efficient deployment a major challenge. Quantization reduces inference memory and computation by representing weights and activations at lower precision. Compared with quantization-aware training (QAT), which requires costly retraining of large models, post-training quantization (PTQ) achieves efficient low-bit deployment using only a small calibration set and without updating model parameters. Building on this practical advantage, recent PTQ methods have improved low-bit inference through complementary strategies, including second-order error compensation, activation-aware rescaling, learned calibration and reconstruction, and more expressive numerical formats (Frantar et al., 2022; Jang & Tambe, 2025; Lin et al., 2024b; Liu et al., 2023; Shao et al., 2024; Xiao et al., 2023; Zou et al., 2024).

Despite this progress, existing PTQ methods still rely on predefined numerical formats with fixed representable-level distributions, limiting their ability to adapt to the diverse value distributions of LLM weights and activations. This mismatch is mainly mitigated in two ways. Transformation- and calibration-based approaches reshape tensor distributions or optimize quantization parameters within a fixed numerical format (Lin et al., 2024b; Shao et al., 2024; Xiao et al., 2023), while format-oriented methods increase representational flexibility through more expressive but still predefined numerical structures (Jang & Tambe, 2025; Liu et al., 2023; Zou et al., 2024). However, their representable levels remain structurally constrained, leading to substantial accuracy degradation at low precision. For example, CBQ (Ding et al., 2025), which optimizes quantization parameters through cross-block reconstruction within a fixed uniform format, still incurs up to a 14.28% drop in zero-shot accuracy under W4A4 quantization.

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

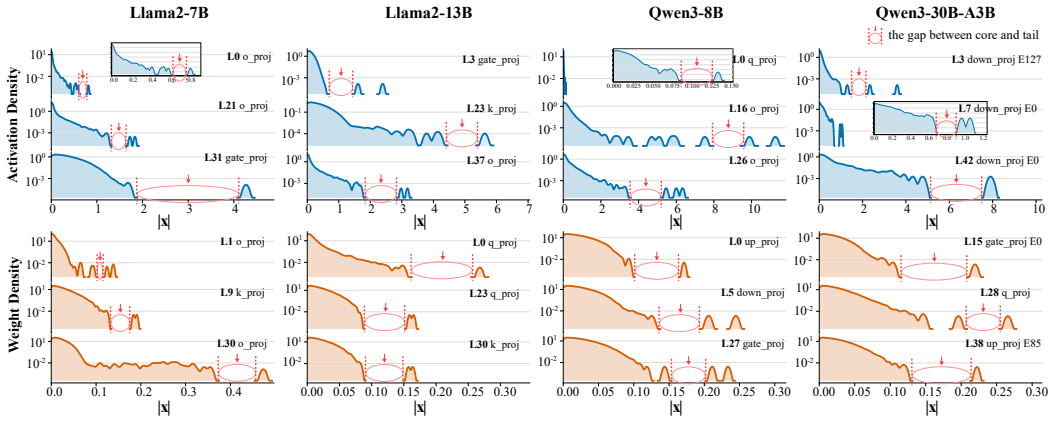

Figure 1: Core-tail structure in representative LLM tensors. Columns show Llama2-7B, Llama2-13B, Qwen3-8B, and Qwen3-30B-A3B. The upper and lower three rows show sampled activation and weight magnitude densities, respectively, on a logarithmic scale. Most values concentrate in a dense core, while a small fraction forms a sparse tail beyond an apparent low-density gap.

Through an in-depth analysis of representative weight and activation tensors from the Llama2 and Qwen3 model families, we uncover a consistent core–tail distributional structure. As illustrated in Figure 1, most values concentrate in a compact, high-density core, while a small fraction of outliers form a sparse tail spanning a much wider magnitude range. The two regions are typically separated by a sharp magnitude transition and a nearly unoccupied interval. This structure consistently appears across model families, tensor types, layers, and projections, yet its boundary and tail range vary substantially across tensors. Consequently, existing approaches built on predefined numerical formats cannot readily adapt to such diverse core–tail distributions, resulting in substantial quantization error at low precision.

Based on these observations, we propose AdaPivot, a new adaptive low-bit numerical format that assigns the dense core and sparse tail separate representational regions, allowing their ranges and resolutions to adapt to tensor-specific distributions through a compact set of learnable discrete and continuous parameters. This design concentrates representational precision on the high-density core while retaining sufficient range for sparse large-magnitude values, avoiding the rigid level distributions of predefined formats. To fit AdaPivot efficiently, we formulate quantization as a joint discrete–continuous reconstruction problem and develop a tailored optimization procedure that co-optimizes the discrete level allocation and the continuous format parameters under an importance-weighted reconstruction error. We further introduce an integer-constrained variant that ties the tail parameters to a shared scale, substantially reducing metadata overhead while preserving the adaptive structure of the format.

In summary, our contributions can be summarized as follows:

- We uncover a consistent core–tail distributional structure in representative LLM weights and activations, and introduce an adaptive low-bit numerical format that assigns the dense core and sparse tail separate representational regions to match tensor-specific distributions.
- We formulate format fitting as a joint discrete–continuous reconstruction problem and develop a tailored optimization procedure, together with an integer-constrained variant that preserves adaptivity while reducing metadata overhead.
- We extensively evaluate AdaPivot across diverse models, tasks, and quantization settings. Under the challenging W4A4 setting, AdaPivot consistently outperforms state-of-the-art PTQ methods, with an average improvement of 7.6% across downstream tasks. Under W4A8, AdaPivot achieves performance comparable to FP16 baselines.

2 Related Work

Weight-only post-training quantization. Weight-only PTQ reduces the memory cost of LLM inference while retaining higher-precision activations. GPTQ uses approximate second-order information

and sequential error compensation to quantize weights (Frantar et al., 2022), whereas AWQ uses activation-aware scaling to protect salient weight channels (Lin et al., 2024b). QuIP combines incoherence processing with adaptive rounding (Chee et al., 2023). Other methods treat difficult weights separately: OWQ retains sensitive weights at higher precision (Lee et al., 2024), SpQR isolates outlier weights in a sparse high-precision representation (Dettmers et al., 2023), and SqueezeLLM combines sensitivity-aware non-uniform quantization with dense-and-sparse decomposition (Kim et al., 2024). Complementary work such as LUT-GEMM focuses on efficient execution of low-bit weight-only matrix multiplication (Park et al., 2024).

Weight-and-activation quantization. Quantizing activations in addition to weights can reduce inference computation, but activation outliers and variation across channels make aggressive low-bit settings more challenging. ZeroQuant introduced fine-grained quantization for large Transformers (Yao et al., 2022), while SmoothQuant transfers part of the activation quantization difficulty to weights through equivalent scaling (Xiao et al., 2023). RPTQ groups channels with similar activation ranges (Yuan et al., 2023), and QLLM redistributes activation outliers through channel reassembly (Liu et al., 2024). OmniQuant learns equivalent transformations and weight clipping under a reconstruction objective (Shao et al., 2024), whereas CBQ extends reconstruction across Transformer blocks to reduce accumulated quantization error (Ding et al., 2025).

Another line of work reshapes the distributions presented to low-bit quantizers. QuaRot applies output-preserving rotations to suppress outliers (Ashkboos et al., 2024b); SpinQuant learns such rotations (Liu et al., 2025); and DuQuant and FlatQuant use more flexible transformations to redistribute or flatten outliers (Lin et al., 2024a; Sun et al., 2025). QUIK and Atom instead use mixed-precision treatments that protect a small set of difficult values or channels during low-bit inference (Ashkboos et al., 2024a; Zhao et al., 2024). These methods primarily mitigate quantization error by changing the input distribution, protecting outliers, or improving reconstruction.

Adaptive numerical representations. Numerical-format design provides another way to accommodate non-uniform distributions in low-bit PTQ. LLM-FP4 searches floating-point format parameters and clipping ranges (Liu et al., 2023), while AFPQ introduces separate scales for positive and negative weights (Zhang et al., 2024a). BiE uses a bi-exponent block floating-point representation (Zou et al., 2024), and BlockDialect selects block-specific formats from a formatbook (Jang & Tambe, 2025). RaZeR remaps the redundant negative-zero encoding to an additional representable value (Chen et al., 2025).

3 METHOD

In this section, we introduce AdaPivot, an adaptive low-bit representation tailored to the dense-core–sparse-tail distributions of LLM weights and activations. Rather than adopting a fixed reconstruction-level layout, AdaPivot jointly adapts the core–tail partition and reconstruction parameters to the observed data distribution, enabling flexible range allocation under a compact representation.

3.1 ADAPTIVE CORE–TAIL NUMERICAL FORMAT

As shown in Figure 2, AdaPivot is an adaptive low-bit numerical format that divides the magnitude range into a dense core and a sparse tail, with the two regions parameterized and adapted independently. Concretely, a discrete allocation variable determines how many levels are assigned to the core, and three continuous parameters determine the core resolution, the position of the first tail level, and the step size between successive tail levels.

Consider a group of values $\boldsymbol{x} = (x_i)_{i=1}^{n}$ to be quantized using B bits, including one sign bit. Since the core–tail allocation depends only on magnitude, we define the format over nonnegative magnitude levels and handle the sign separately through sign-magnitude encoding. Let $M = 2^{B-1}$ denote the number of nonnegative magnitude indices. We assign the first C magnitude indices, $k = 0, \ldots, C-1$, to the core, where $k = 0$ represents zero, and the remaining $T = M - C$ indices to the tail, with $1 \le C < M$. With continuous parameters $\boldsymbol{\theta} = (s_c, a, s_t)$, the nonnegative magnitude levels are:

$$\mathcal{V}(C, \boldsymbol{\theta}) = \{k s_c \mid k = 0, \ldots, C-1\} \cup \{a + j s_t \mid j = 0, \ldots, T-1\}, \tag{1}$$

where s_c and s_t are the step sizes of the core and tail segments, respectively, while a specifies the first tail reconstruction level. For a fixed allocation C, the continuous parameters are positive and

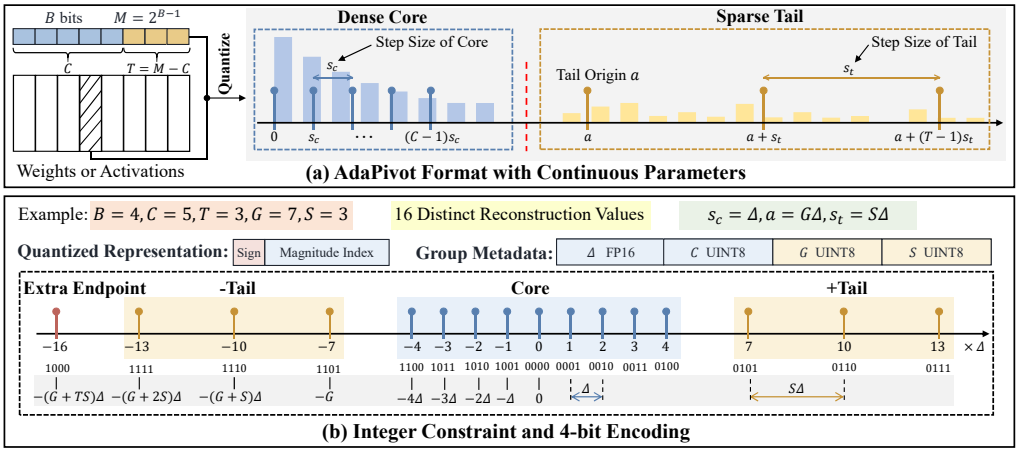

Figure 2: Adaptive core–tail format and its compact integer-constrained encoding. (a) Continuous form with independently parameterized core spacing, tail origin, and tail spacing. (b) A 4-bit example of the integer-constrained form, where $s_c = \Delta, a = G\Delta$ and $s_t = S\Delta$.

satisfy $a \geq Cs_c$, so that the levels increase monotonically within each segment and the two segments do not overlap. We denote the resulting feasible parameter set by Ω_C. The allocation C and the three continuous parameters are jointly optimized by the fitting procedure described in Section 3.2.

A naive sign-magnitude extension mirrors the magnitude levels on the negative side, but this wastes one code: zero is represented twice, as $+0$ and -0. To eliminate this redundancy, we constrain the negative-zero code to represent an additional negative tail endpoint $-(a + Ts_t)$, extending the tail range in the direction where large-magnitude outliers concentrate. This yields the signed quantization numerical format $\mathcal{K}(C, \boldsymbol{\theta})$ and the associated nearest-level quantizer:

$$\mathcal{K}(C, \boldsymbol{\theta}) = \{0\} \cup \{\pm v \mid v \in \mathcal{V}(C, \boldsymbol{\theta}), v > 0\} \cup \{-(a + Ts_t)\}, \tag{2}$$

$$Q_{C,\boldsymbol{\theta}}(x) = \operatorname*{argmin}_{v \in \mathcal{K}(C,\boldsymbol{\theta})} (x - v)^2. \tag{3}$$

This construction utilizes all 2^B codes, resolving the positive- and negative-zero redundancy of sign-magnitude encoding. For example, with $B = 4$, signed code 1000 represents the extra negative endpoint, while the remaining codes follow the standard sign and magnitude-index interpretation.

3.2 Joint Discrete–Continuous Format Fitting

Figure 3 presents an overview of the fitting procedure. Given a quantization group $\boldsymbol{x}$, AdaPivot fits the discrete core–tail allocation and the continuous format parameters by minimizing the weighted reconstruction error:

$$\min_{C \in \mathcal{C},\, \boldsymbol{\theta} \in \Omega_C} \mathcal{L}_g = \sum_i \alpha_i \left[x_i - Q_{C,\boldsymbol{\theta}}(x_i)\right]^2, \tag{4}$$

where $\mathcal{C}$ is the candidate set of core allocations (i.e., the number of core levels) and $\alpha_i \geq 0$ is the importance weight of x_i. For activation groups, $\alpha_i = \sum_m \|(W_m)_{:,i}\|_2^2$ aggregates the effect of input channel i on all weight matrices W_m that consume the same activation; for weight groups, α_i is derived from the Hessian of the layer-wise weight reconstruction objective. The discrete variable C determines how many magnitude levels are assigned to the core, and hence how many remain for the tail, while $\boldsymbol{\theta}$ controls their resolution and placement. Because nearest-level assignments depend on $\boldsymbol{\theta}$, the objective is non-convex even when C is fixed; we therefore optimize it by alternating between level assignment and parameter updates, each subproblem can be solved efficiently when the other is fixed.

For each candidate allocation $C \in \mathcal{C}$, we optimize $\boldsymbol{\theta}$ by alternating between nearest-level assignment and parameter update. Given $\boldsymbol{\theta}$, each x_i is assigned to its nearest level in $\mathcal{K}(C, \boldsymbol{\theta})$ according to the

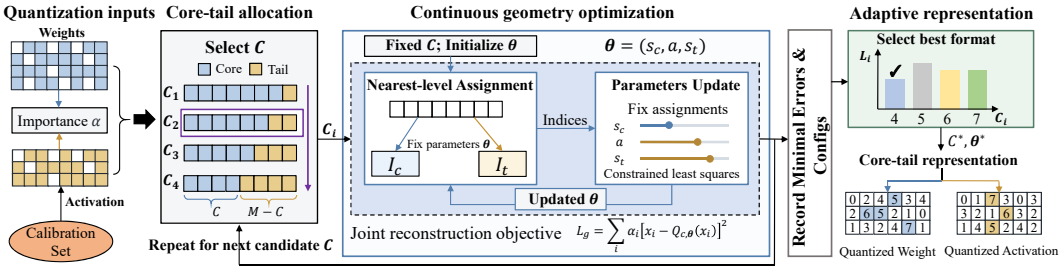

Figure 3: Overview of AdaPivot. For each weight or activation group, AdaPivot selects the core–tail allocation C from a predefined candidate set and adapts the continuous format parameters $\theta = (s_c, a, s_t)$. With C fixed, nearest-level assignment and constrained least-squares parameter updates alternate to minimize importance-weighted reconstruction error. The best fitted candidate defines the adaptive core–tail format.

quantizer in Eq. 3. This assignment partitions the group into two index sets: $\mathcal{I}_c$ and $\mathcal{I}_t$, containing the entries assigned to the core and tail levels, respectively. For $i \in \mathcal{I}_c$, we denote by $k_i \in \{0, \ldots, C-1\}$ the index of the assigned core level; for $i \in \mathcal{I}_t$, we denote by $j_i \in \{0, \ldots, T-1\}$ the index of the assigned tail level, with $j_i = T$ reserved for entries assigned to the extra negative endpoint $-(a + Ts_t)$. Let $y_i = |x_i|$. With the assignments fixed, the parameter update becomes:

$$\min_{\boldsymbol{\theta} \in \Omega_C} \sum_{i \in \mathcal{I}_c} \alpha_i (y_i - k_i s_c)^2 + \sum_{i \in \mathcal{I}_t} \alpha_i (y_i - a - j_i s_t)^2. \tag{5}$$

Eq. 5 is quadratic in $\boldsymbol{\theta}$ with linear constraints, and hence a convex quadratic program. When the constraints are inactive and $\sum_{i \in \mathcal{I}_c} \alpha_i {k_i}^2 > 0$, it admits closed-form solutions, with the convexity argument and derivations provided in Appendix A. The optimal core step size is:

$${s_c}^* = \frac{\sum_{i \in \mathcal{I}_c} \alpha_i k_i y_i}{\sum_{i \in \mathcal{I}_c} \alpha_i {k_i}^2}, \tag{6}$$

and, with the weighted moments $u_r = \sum_{i \in \mathcal{I}_t} \alpha_i {j_i}^r$ $(r \in \{0,1,2\})$, $v_r = \sum_{i \in \mathcal{I}_t} \alpha_i {j_i}^r y_i$ $(r \in \{0,1\})$, and $\delta = u_0 u_2 - u_1^2 > 0$, the optimal tail parameters are:

$$a^* = \frac{u_2 v_0 - u_1 v_1}{\delta}, \qquad {s_t}^* = \frac{u_0 v_1 - u_1 v_0}{\delta}. \tag{7}$$

If the unconstrained estimates violate the feasibility conditions—for example, a non-positive step size or a tail origin $a < (C-1)\, s_c$ that overlaps the core segment—we fall back to solving the constrained quadratic program in Eq. 5 directly. Degenerate assignments occasionally arise: if no entry is assigned to a segment, its step size drops out of the objective, and if all tail entries share a single index j, only the combination $a + js_t$ is identified rather than a and s_t individually. In such cases some parameters are unidentifiable and Eq. 5 admits multiple minimizers; we simply adopt any feasible one, which does not affect the final result since all feasible minimizers attain the same objective value.

The alternation terminates when the assignments stabilize; since each update does not increase the group loss, the objective decreases monotonically within each run. For the current C, we record the fitted parameters and the corresponding loss. After all candidate allocations have been evaluated, the pair with the lowest loss is selected as the final solution $(C^*, \boldsymbol{\theta}^*)$.

Integer-constrained variant. The continuous formulation above represents the most general form of AdaPivot; however, storing s_c, a, and s_t as three independent floating-point parameters per group incurs considerable metadata overhead, especially for small quantization groups. To reduce this overhead, we tie the tail parameters to the core step size through integer ratios:

$$s_c = \Delta, \quad a = G\Delta, \quad s_t = S\Delta, \quad \Delta \geq \epsilon, \quad G \in \{C, \ldots, G_{\max}\}, \quad S \in \mathcal{S} \subset \mathbb{Z}_{>0}. \tag{8}$$

The normalized nonnegative levels become $\{0, \ldots, C-1\} \cup \{G + jS : j = 0, \ldots, T-1\}$, where G is the integer tail origin in units of Δ and S is the tail-to-core step-size ratio; the format remains adaptive through the group-specific parameters (C, G, S, Δ). Fitting then reduces to a small discrete search over (C, G, S) with a closed-form scale update: for fixed assignments, let z_i denote the assigned normalized magnitude ($z_i = k_i$ in the core and $z_i = G + Sj_i$ in the tail); the conditional scale update then admits the closed form:

$$\Delta^* = \frac{\sum_i \alpha_i z_i y_i}{\sum_i \alpha_i z_i^2}, \tag{9}$$

and G is updated by rounding the weighted tail mean of $y_i/\Delta - Sj_i$ to nearby integers. This variant stores one FP16 scale Δ and one byte each for C, G, and S—40 metadata bits per group, versus 56 bits for the continuous form—amounting to $B + \frac{40}{n}$ bits per value for a group of n values. Notably, AdaPivot subsumes the conventional uniform format as a special case: for $B = 4$, the choice $C = 7$, $G = 7$, and $S = 1$ recovers the signed uniform format $\{-8, \ldots, 7\}\Delta$, whereas other choices of (C, G, S) realize non-uniform level layouts that uniform quantization cannot represent.

3.3 Integration into Weight–Activation PTQ

We integrate AdaPivot into a standard weight–activation PTQ pipeline with layer-wise output reconstruction, proceeding in two stages. First, we calibrate the activation format offline: the discrete parameters (C, G, S) are selected on the calibration set, while a per token scale is computed at runtime, so inference only recomputes this scale. Second, with the activation format fixed, we quantize the weights and adopt the GPTQ (Frantar et al., 2022) procedure to propagate the weight quantization error, which compensates for the error introduced by already-quantized weights and reduces the overall quantization error. The output of each fully quantized layer is propagated as the calibration input to the next layer. When attention-operand quantization is enabled, the same activation quantization process is applied independently to the post-RoPE query, key, and value tensors. Despite the additional per-group format search, the calibration overhead remains modest: the entire quantization procedure typically completes within two hours on a single NVIDIA H20 GPU. The complete calibration and fitting order is summarized in Appendix B.

4 Experiments

4.1 Experimental Setup

Models and Evaluation. We primarily evaluate AdaPivot on the LLaMA-2 (Touvron et al., 2023) and Qwen3 (Yang et al., 2025) model families, covering Llama2-7B, Llama2-13B, Llama2-70B, Qwen3-8B, and Qwen3-30B-A3B (denoted Qwen3-30B). Language modeling is measured by token perplexity on WikiText-2 (Merity et al., 2016) and C4 (Raffel et al., 2020). For commonsense reasoning, we evaluate on five zero-shot tasks: PIQA (Bisk et al., 2020), ARC-Easy and ARC-Challenge (Clark et al., 2018), HellaSwag (Zellers et al., 2019), and WinoGrande (Sakaguchi et al., 2021), using normalized accuracy for the first four tasks and accuracy for WinoGrande.

Quantization settings. Our main experiments use W4A4 and W4A8, with weights quantized per output channel and activations (including KV) per token, while retaining embeddings and the language-model head in their original precision. A supplementary Llama2-13B experiment evaluates W4A16 weight-only quantization with a group size of 128.

Baseline methods. We compare AdaPivot with SmoothQuant (Xiao et al., 2023), OmniQuant (Shao et al., 2024), CBQ (Ding et al., 2025) and SliderQuant (Wang et al., 2026) under W4A4, and with SmoothQuant, OmniQuant, QQQ (Zhang et al., 2024b), and QoQ (Lin et al., 2025) under W4A8. The FP16 model serves as the original baseline. SmoothQuant is evaluated through its low-bit extension at the corresponding precision. The W4A16 comparison includes RTN-MinMax, RTN-Clip, GPTQ, and AWQ. All methods adopt the same downstream tasks and perplexity evaluation protocol, while retaining their original method-specific calibration procedures.

Table 1: Main W4A4 results on Llama2 and Qwen3. Wiki2 denotes WikiText-2; PPL is lower-is-better; Avg. is the mean accuracy (%) over the five downstream tasks.

Model	Method	W/A	Wiki2 PPL ↓	C4 PPL ↓	PIQA ↑	ARC-E ↑	ARC-C ↑	Hella. ↑	Wino. ↑	Avg. ↑
Llama2-7B	FP16	16/16	5.47	6.97	78.51	73.95	44.97	76.16	69.77	68.67
	SmoothQuant	4/4	83.12	77.27	60.17	35.23	27.13	37.08	49.57	41.84
	OmniQuant	4/4	14.16	18.02	65.94	43.94	30.80	53.53	55.09	49.86
	CBQ	4/4	12.73	14.45	70.18	56.40	33.45	60.46	55.09	55.12
	SliderQuant	4/4	**8.34**	**11.10**	71.38	59.64	32.94	**62.33**	**61.72**	57.60
	AdaPivot	**4/4**	10.45	13.42	**71.60**	**60.75**	**35.75**	61.96	58.48	**57.71**
Llama2-13B	FP16	16/16	4.88	6.47	80.25	76.43	48.81	77.32	72.77	71.12
	SmoothQuant	4/4	46.62	43.19	62.30	40.28	30.72	42.24	49.96	45.10
	OmniQuant	4/4	12.30	14.55	69.80	47.22	33.79	59.34	55.49	53.13
	CBQ	4/4	8.48	11.71	71.00	61.57	35.84	65.15	57.93	58.30
	SliderQuant	4/4	7.62	10.26	71.65	62.88	37.80	66.02	60.77	59.82
	AdaPivot	**4/4**	**7.15**	**9.58**	**74.54**	**70.16**	**42.58**	**66.80**	**66.38**	**64.09**
Llama2-70B	FP16	16/16	3.32	5.52	82.37	79.55	57.25	82.13	80.03	76.27
	SmoothQuant	4/4	33.40	43.28	61.37	46.24	31.23	52.65	50.91	48.47
	OmniQuant	4/4	11.54	13.72	71.71	59.55	37.20	66.63	58.17	58.65
	CBQ	4/4	7.56	11.04	73.24	62.45	37.30	69.84	61.23	60.81
	SliderQuant	4/4	6.87	9.67	75.79	64.14	37.71	73.02	65.75	63.28
	AdaPivot	**4/4**	**4.90**	**7.31**	**78.78**	**76.05**	**51.54**	**76.26**	**71.19**	**70.76**
Qwen3-8B	FP16	16/16	9.71	13.29	77.80	80.93	56.40	74.93	67.80	71.57
	SmoothQuant	4/4	4401.93	4133.25	51.85	27.57	25.00	26.03	50.75	36.24
	OmniQuant	4/4	171.53	184.50	58.00	39.81	27.39	30.36	50.51	41.21
	AdaPivot	**4/4**	**13.43**	**18.39**	**69.53**	**66.67**	**44.80**	**63.13**	**62.90**	**61.41**
Qwen3-30B-A3B	FP16	16/16	8.71	12.08	80.14	79.25	56.23	77.66	69.93	72.64
	SmoothQuant	4/4	34.27	36.90	63.55	57.70	36.77	53.11	55.49	53.32
	OmniQuant	4/4	52.43	41.98	67.41	55.64	33.28	52.34	52.09	52.15
	AdaPivot	**4/4**	**13.60**	**17.55**	**68.99**	**61.11**	**43.34**	**63.20**	**60.62**	**59.45**

Table 2: Main W4A8 results on Llama2-7b and 13b.

Model	Method	W/A	Wiki2 PPL ↓	C4 PPL ↓	PIQA ↑	ARC-E ↑	ARC-C ↑	Hella. ↑	Wino. ↑	Avg. ↑
Llama2-7b	FP16	16/16	5.47	6.97	78.51	73.95	44.97	76.16	69.77	68.67
	SmoothQuant	4/8	7.55	9.30	76.66	66.41	40.79	70.27	64.88	63.80
	OmniQuant	4/8	5.76	7.36	78.13	70.41	42.24	74.21	68.43	66.68
	QQQ	4/8	5.95	–	77.42	69.15	42.15	73.54	65.98	65.65
	QoQ	4/8	5.75	–	77.64	72.81	43.60	74.00	68.03	67.22
	AdaPivot	**4/8**	**5.61**	**7.20**	**78.24**	**73.27**	**43.69**	**74.36**	**69.53**	**67.82**
Llama2-13b	FP16	16/16	4.88	6.47	80.25	76.43	48.81	77.32	72.77	71.12
	SmoothQuant	4/8	5.99	7.64	77.80	75.00	46.76	71.07	69.46	68.02
	OmniQuant	4/8	5.07	6.71	79.71	75.72	47.44	**78.49**	**72.69**	70.81
	QQQ	4/8	5.21	–	79.43	74.75	48.12	77.27	70.32	69.98
	QoQ	4/8	5.12	–	79.71	75.97	48.38	77.80	70.96	70.56
	AdaPivot	**4/8**	**5.00**	**6.62**	**80.41**	**76.47**	**49.40**	76.50	72.53	**71.06**

Implementation details. Following previous works (Frantar et al., 2022; Shao et al., 2024), we randomly sample 128 sequences of 2048 tokens from the WikiText-2 training set as the calibration data, using random seed 2 for all Llama2 and Qwen3 weight-activation runs. For 4-bit weights, we search C in $\{4, 5, 6, 7\}$ and S in $\{1, 2, 3, 4, 6, 8\}$, using three fitting iterations and a pivot radius of one. Evaluation uses a sequence length of 2048; C4 uses 256 validation sequences with seed 0. All five downstream tasks are evaluated without a sample limit.

4.2 Main Results

W4A4. Table 1 compares AdaPivot with representative weight-activation PTQ methods under W4A4. On Llama2, AdaPivot consistently achieves the best overall performance at 13B and 70B, and the margin over the strongest baseline grows with model scale, reaching 7.5 percentage points of average zero-shot accuracy and a 28.7% WikiText-2 perplexity reduction at 70B. The advantage is more pronounced on Qwen3, which is considerably harder to quantize: SmoothQuant collapses on Qwen3-8B and OmniQuant remains far from the FP16 baselines, whereas AdaPivot reduces

Table 3: Overall ablation on Llama2-13B W4A4. Adaptive denotes parameters determined using the proposed optimization method.

Variant	C	G	S	WikiText-2 PPL ↓	C4 PPL ↓
Uniform	7	7	1	127.50	212.42
Unit tail stride	Adaptive	Adaptive	1	48.86	91.36
No core-tail gap	Adaptive	C	Adaptive	7.55	10.03
Full AdaPivot	Adaptive	Adaptive	Adaptive	7.15	9.58

Table 4: Fixed versus adaptive core allocation. G, S, and scale fitting follow the original procedure.

Core allocation	WikiText-2 PPL ↓	C4 PPL ↓
Adaptive C (Full)	**7.15**	**9.58**
C = 4	7.23	9.59
C = 5	7.89	11.04
C = 6	11.29	19.31
C = 7	67.69	150.80

Table 5: Fixed versus adaptive tail stride.

Tail stride	WikiText-2 PPL ↓	C4 PPL ↓
Adaptive S (Full)	**7.15**	**9.58**
S = 1	48.86	91.36
S = 2	13.35	20.04
S = 4	7.49	9.63
S = 6	7.37	9.68
S = 8	9.24	13.24

the perplexity by an order of magnitude and improves average accuracy by up to 20 percentage points, with the same trend holding for the MoE model Qwen3-30B-A3B. Both trends are consistent with our motivation: larger models and the Qwen3 family exhibit heavier activation tails with a sharper core–tail separation, which predefined numerical formats with fixed level distributions cannot accommodate—transferring the quantization difficulty to weights or optimizing within a fixed format does not resolve this mismatch. AdaPivot instead reallocates levels between the core and the tail on a per-tensor basis, and therefore benefits more as the tail becomes more pronounced.

W4A8. Table 2 reports W4A8 results on Llama2-7B and 13B. With activation error largely removed, AdaPivot nearly closes the gap to FP16, with the average accuracy at both model sizes matching the FP16 baselines. AdaPivot also outperforms the baselines in both perplexity and accuracy, indicating that an adaptive format remains beneficial even when activations no longer dominate the error.

Appendix Table S1 provides the additional W4A16 comparison on Llama2-13B.

4.3 Ablation Studies

To better understand the contribution of each design component in AdaPivot, we conduct a series of ablation studies and analyze their individual effects on quantization performance.

Overall Ablation. Table 3 shows how each design progressively improves AdaPivot. Starting from the uniform configuration, enabling adaptive core–tail allocation reduces WikiText-2 PPL from 127.50 to 48.86. Further adapting the tail stride lowers it to 7.55, demonstrating the importance of matching the sparse tail with flexible spacing. Finally, introducing an explicit gap between the core and tail further reduces WikiText-2 PPL to 7.15 and C4 PPL to 9.58. These results show that the three designs contribute complementary gains toward accurately matching the underlying tensor distribution.

Effect of Adaptive Optimization for C and S. Tables 4 and 5 evaluate fixed core allocations and tail strides against their adaptively determined counterparts. For C, a small fixed allocation can approach the full method, but performance degrades rapidly as more levels are assigned to the core, indicating that excessive core allocation leaves insufficient capacity for representing the sparse tail. A similar sensitivity is observed for S: moderate fixed strides remain competitive, whereas overly small or large values incur substantial degradation, with $S = 1$ increasing WikiText-2 PPL from 7.15 to 48.86. Overall, jointly determining C and S through the proposed optimization avoids manual parameter selection and consistently provides a better balance between core resolution and tail coverage.

Effect of Extra-Code Placement. In AdaPivot, the additional code obtained by reusing the negative-zero encoding is assigned to the negative tail endpoint. Table 6 compares this design with placing

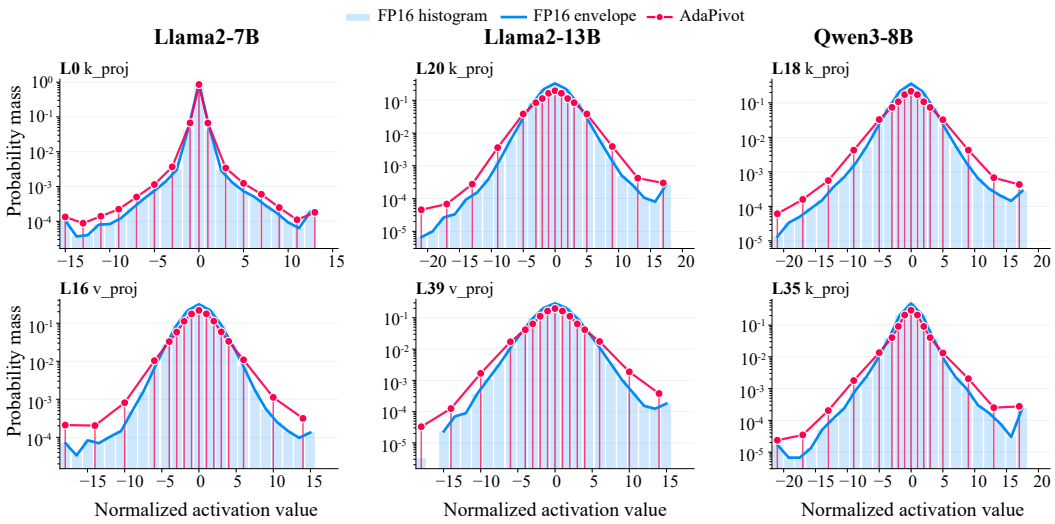

Figure 4: FP16 activation distributions and AdaPivot W4A4 code allocation across models and layers. Blue bars show the histogram of FP16 values, with the blue curve tracing the corresponding distribution, while the red curve represents the AdaPivot quantized distribution.

Table 6: Effect of extra code assignment to the positive or negative tail on Llama2-13b.

Method	W/A	Wiki2 PPL ↓	C4 PPL ↓	PIQA ↑	ARC-E ↑	ARC-C ↑	Hella. ↑	Wino. ↑	Avg. ↑
FP16	16/16	4.88	6.47	80.25	76.43	48.81	77.32	72.77	71.12
negative outlier	4/4	7.15	**9.58**	74.54	**70.16**	**42.58**	66.80	**66.38**	**64.09**
positive outlier	4/4	**7.04**	9.67	**75.35**	68.27	40.96	**67.14**	62.67	62.88

the extra code on the positive tail. Assigning the extra code to the negative outlier region yields consistently better downstream performance, while the positive-side variant offers no clear advantage even in perplexity. We thus assign the extra code to the negative tail by default.

4.4 Quantized Distribution Visualization

As shown in Figure 4, to examine how well AdaPivot preserves the original value distributions after quantization, we visualize representative layers from Llama2-7B, Llama2-13B, and Qwen3-8B. Across different models and layers, the quantized distributions closely match the original ones, faithfully preserving both the concentrated core and the sparse tail. This consistent agreement demonstrates that AdaPivot can effectively capture diverse tensor-specific distributional characteristics rather than imposing a fixed representational geometry.

5 Conclusion

We introduced AdaPivot, an adaptive core-tail representation for low-bit LLM quantization. Unlike fixed numerical formats, AdaPivot treats level allocation, core resolution, tail origin, and tail spacing as optimization variables, allowing the reconstruction parameters to match tensor-specific distributions. Its joint discrete-continuous formulation provides a principled fitting objective, while the integer-constrained specialization retains the adaptive structure with compact metadata. AdaPivot consistently improves perplexity and zero-shot accuracy across Llama2 and Qwen3 models. It substantially narrows the gap to full precision under W4A4 quantization and closely matches FP16 performance under W4A8. Overall, the results establish adaptive reconstruction-level parameters as an effective design principle for accurate low-bit quantization.

AI USE STATEMENT

In this work, we used generative AI tools for editing the manuscript to improve readability, including language polishing and grammar and spelling checks. We have not used generative AI tools for any tasks requiring disclosure, such as generating synthetic data, developing theoretical models, formulating or proving mathematical claims, designing experiments, or implementing methods. We have reviewed all AI-assisted output, and we take full responsibility for the final content of this work, including text, claims, and artifacts produced with the aid of generative AI.

REPRODUCIBILITY STATEMENT

We provide an anonymous code repository to facilitate the reproduction of our results. The release includes the complete implementation of AdaPivot, covering format fitting, model integration, and baseline adapters, together with the full experiment configurations that specify the calibration dataset, sample count, sequence length, and random seed for every reported result. The format definition and fitting procedure are described in Section 3, with the derivations of the closed-form updates given in Appendix A and the complete calibration and fitting order summarized in Appendix B. All evaluation datasets and protocols are described in the experimental setup, and the configurations and integrity manifests for the ablation studies in Section 4.3 are also included in the release.

REFERENCES

Josh Achiam, Steven Adler, Sandhini Agarwal, Lama Ahmad, Ilge Akkaya, Florencia Leoni Aleman, Diogo Almeida, Janko Altenschmidt, Sam Altman, Shyamal Anadkat, et al. GPT-4 technical report. *arXiv preprint arXiv:2303.08774*, 2023.

Saleh Ashkboos, Ilia Markov, Elias Frantar, Tingxuan Zhong, Xincheng Wang, Jie Ren, Torsten Hoefler, and Dan Alistarh. Quik: Towards end-to-end 4-bit inference on generative large language models. In *Proceedings of the 2024 Conference on Empirical Methods in Natural Language Processing*, pp. 3355–3371, 2024a.

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

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

Tom Brown, Benjamin Mann, Nick Ryder, Melanie Subbiah, Jared D Kaplan, Prafulla Dhariwal, Arvind Neelakantan, Pranav Shyam, Girish Sastry, Amanda Askell, et al. Language models are few-shot learners. *Advances in neural information processing systems*, 33:1877–1901, 2020.

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

Yuzong Chen, Xilai Dai, Jake Hyun, Chi-Chih Chang, Wonsuk Jang, Yuheng Wu, Thierry Tambe, Jae-sun Seo, and Mohamed S Abdelfattah. Razer: Pushing the limits of nvfp4 quantization with redundant zero remapping. *arXiv preprint arXiv:2501.04052*, 2025.

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

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

Xin Ding, Xiaoyu Liu, Zhijun Tu, Yun Zhang, Wei Li, Jie Hu, Hanting Chen, Yehui Tang, Zhiwei Xiong, Baoqun Yin, et al. Cbq: Cross-block quantization for large language models. In *International Conference on Learning Representations*, volume 2025, pp. 7056–7075, 2025.

Elias Frantar, Saleh Ashkboos, Torsten Hoefler, and Dan Alistarh. Gptq: Accurate post-training quantization for generative pre-trained transformers. *arXiv preprint arXiv:2210.17323*, 2022.

Wonsuk Jang and Thierry Tambe. Blockdialect: block-wise fine-grained mixed format quantization for energy-efficient llm inference. In *Proceedings of the 42nd International Conference on Machine Learning*, ICML'25. JMLR.org, 2025.

Sehoon Kim, Coleman Richard Charles Hooper, Amir Gholami, Zhen Dong, Xiuyu Li, Sheng Shen, Michael W. Mahoney, and Kurt Keutzer. SqueezeLLM: Dense-and-sparse quantization. In *Forty-first International Conference on Machine Learning*, 2024. URL https://openreview.net/forum?id=0jpbpFia8m.

Changhun Lee, Jungyu Jin, Taesu Kim, Hyungjun Kim, and Eunhyeok Park. Owq: Outlier-aware weight quantization for efficient fine-tuning and inference of large language models. In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 38, pp. 13355–13364, 2024.

Haokun Lin, Haobo Xu, Yichen Wu, Jingzhi Cui, Yingtao Zhang, Linzhan Mou, Linqi Song, Zhenan Sun, and Ying Wei. Duquant: Distributing outliers via dual transformation makes stronger quantized llms. *Advances in Neural Information Processing Systems*, 37:87766–87800, 2024a.

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

Yujun Lin, Haotian Tang, Shang Yang, Zhekai Zhang, Guangxuan Xiao, Chuang Gan, and Song Han. Qserve: W4a8kv4 quantization and system co-design for efficient llm serving. *Proceedings of Machine Learning and Systems*, 7, 2025.

Jing Liu, Ruihao Gong, Xiuying Wei, Zhiwei Dong, Jianfei Cai, and Bohan Zhuang. Qllm: Accurate and efficient low-bitwidth quantization for large language models. In *International Conference on Learning Representations*, volume 2024, pp. 15417–15439, 2024.

Shih-yang Liu, Zechun Liu, Xijie Huang, Pingcheng Dong, and Kwang-Ting Cheng. Llm-fp4: 4-bit floating-point quantized transformers. In *Proceedings of the 2023 conference on empirical methods in natural language processing*, pp. 592–605, 2023.

Zechun Liu, Changsheng Zhao, Igor Fedorov, Bilge Soran, Dhruv Choudhary, Raghuraman Krishnamoorthi, Vikas Chandra, Yuandong Tian, and Tijmen Blankevoort. Spinquant: Llm quantization with learned rotations. In *International Conference on Learning Representations*, volume 2025, pp. 92009–92032, 2025.

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

Gunho Park, Minsub Kim, Sungjae Lee, Jeonghoon Kim, Beomseok Kwon, Se Jung Kwon, Byeongwook Kim, Youngjoo Lee, Dongsoo Lee, et al. Lut-gemm: Quantized matrix multiplication based on luts for efficient inference in large-scale generative language models. In *International Conference on Learning Representations*, volume 2024, pp. 38069–38086, 2024.

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

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

Wenqi Shao, Mengzhao Chen, Zhaoyang Zhang, Peng Xu, Lirui Zhao, Zhiqian Li, Kaipeng Zhang, Gao Peng, Yu Qiao, and Ping Luo. Omniquant: Omnidirectionally calibrated quantization for large language models. In *International Conference on Learning Representations*, volume 2024, pp. 45472–45496, 2024.

Yuxuan Sun, Ruikang Liu, Haoli Bai, Han Bao, Kang Zhao, Yuening Li, Jiaxin Hu, Xianzhi Yu, Lu Hou, Chun Yuan, Xin Jiang, Wulong Liu, and Jun Yao. Flatquant: Flatness matters for LLM quantization. In *Forty-second International Conference on Machine Learning*, 2025. URL https://openreview.net/forum?id=uTz2Utym5n.

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

Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N Gomez, Łukasz Kaiser, and Illia Polosukhin. Attention is all you need. *Advances in neural information processing systems*, 30, 2017.

Shigeng Wang, Chao Li, Yangyuxuan Kang, Jiawei Fan, Zhonghong Ou, and Anbang Yao. Sliderquant: Accurate post-training quantization for LLMs. In *The Fourteenth International Conference on Learning Representations*, 2026. URL https://openreview.net/forum?id=YNqZqw4fLT.

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

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

Zhewei Yao, Reza Yazdani Aminabadi, Minjia Zhang, Xiaoxia Wu, Conglong Li, and Yuxiong He. Zeroquant: Efficient and affordable post-training quantization for large-scale transformers. *Advances in neural information processing systems*, 35:27168–27183, 2022.

Zhihang Yuan, Lin Niu, Jiawei Liu, Wenyu Liu, Xinggang Wang, Yuzhang Shang, Guangyu Sun, Qiang Wu, Jiaxiang Wu, and Bingzhe Wu. Rptq: Reorder-based post-training quantization for large language models. *arXiv preprint arXiv:2304.01089*, 2023.

Rowan Zellers, Ari Holtzman, Yonatan Bisk, Ali Farhadi, and Yejin Choi. Hellaswag: Can a machine really finish your sentence? In *Proceedings of the 57th annual meeting of the association for computational linguistics*, pp. 4791–4800, 2019.

Yijia Zhang, Sicheng Zhang, Shijie Cao, Dayou Du, Jianyu Wei, Ting Cao, and Ningyi Xu. Afpq: Asymmetric floating point quantization for llms. In *Findings of the Association for Computational Linguistics: ACL 2024*, pp. 28–36, 2024a.

Ying Zhang, Peng Zhang, Mincong Huang, Jingyang Xiang, Yujie Wang, Chao Wang, Yineng Zhang, Lei Yu, Chuan Liu, and Wei Lin. Qqq: Quality quattuor-bit quantization for large language models. *arXiv preprint arXiv:2406.09904*, 2024b.

Yilong Zhao, Chien-Yu Lin, Kan Zhu, Zihao Ye, Lequn Chen, Size Zheng, Luis Ceze, Arvind Krishnamurthy, Tianqi Chen, and Baris Kasikci. Atom: Low-bit quantization for efficient and accurate llm serving. *Proceedings of Machine Learning and Systems*, 6:196–209, 2024.

Lancheng Zou, Wenqian Zhao, Shuo Yin, Chen Bai, Qi Sun, and Bei Yu. Bie: Bi-exponent block floating-point for large language models quantization. In *Forty-first International Conference on Machine Learning*, 2024.

A Derivation of the Continuous Parameters Updates

A.1 Fixed-Assignment Quadratic Program

For a fixed allocation C and fixed nearest-level assignments, define $\boldsymbol{\theta} = [s_c, a, s_t]^T$. For each entry, define the design vector

$$\boldsymbol{\phi}_i = \begin{cases} (k_i, 0, 0)^T, & i \in \mathcal{I}_c, \\ (0, 1, j_i)^T, & i \in \mathcal{I}_t, \end{cases} \tag{A.1}$$

Stacking ϕ_i^T as the rows of $\boldsymbol{\Phi}$, the magnitudes y_i into y, and the importance weights into $W_\alpha = \mathrm{diag}(\alpha_1, \ldots, \alpha_n)$, the fixed-assignment objective can be written as

$$L(\boldsymbol{\theta}) = (\boldsymbol{y} - \boldsymbol{\Phi}\boldsymbol{\theta})^T \boldsymbol{W}_\alpha (\boldsymbol{y} - \boldsymbol{\Phi}\boldsymbol{\theta}). \tag{A.2}$$

Its Hessian is

$$\nabla^2 L(\boldsymbol{\theta}) = 2\boldsymbol{\Phi}^T \boldsymbol{W}_\alpha \boldsymbol{\Phi} \succeq \mathbf{0} \tag{A.3}$$

Because $\alpha_i \geq 0$, the diagonal matrix $\boldsymbol{W}_\alpha$ is positive semidefinite, and hence the Hessian in Eq. A.3 is positive semidefinite. The feasible set Ω_C is defined by linear inequalities. Therefore, Eq. 5 is a convex quadratic program for fixed assignments.

A.2 Core Step-Size Update

The core-dependent part of the objective is

$$L_c(s_c) = \sum_{i \in \mathcal{I}_c} \alpha_i (y_i - k_i s_c)^2 \tag{A.4}$$

Differentiating with respect to s_c gives

$$\frac{\partial \mathcal{L}_c}{\partial s_c} = -2 \sum_{i \in \mathcal{I}_c} \alpha_i k_i (y_i - k_i s_c) \tag{A.5}$$

Setting this derivative to zero yields ${s_c}^* = \frac{\sum_{i \in \mathcal{I}_c} \alpha_i k_i y_i}{\sum_{i \in \mathcal{I}_c} \alpha_i {k_i}^2}$, provided that the denominator is positive.

A.3 Tail Origin and Step-Size Updates

The tail-dependent objective is

$$L_t(a, s_t) = \sum_{i \in \mathcal{I}_t} \alpha_i (y_i - a - j_i s_t)^2 \tag{A.6}$$

Setting the derivatives with respect to a and s_t to zero gives the normal equations

$$\begin{bmatrix} u_0 & u_1 \\ u_1 & u_2 \end{bmatrix} \begin{bmatrix} a \\ s_t \end{bmatrix} = \begin{bmatrix} v_0 \\ v_1 \end{bmatrix} \tag{A.7}$$

The determinant of the coefficient matrix is

$$\delta = u_0 u_2 - u_1^2 \tag{A.8}$$

When $\delta > 0$, solving Eq. A.7 gives

$$a^* = \frac{u_2 v_0 - u_1 v_1}{\delta}, {s_t}^* = \frac{u_0 v_1 - u_1 v_0}{\delta} \tag{A.9}$$

The condition $\delta > 0$ requires positive-weight tail assignments at more than one distinct tail index. If $\delta = 0$, the assignments do not contain enough information to identify a and s_t separately.

Algorithm 1 Layer-wise AdaPivot calibration

Require: Pretrained model F, calibration data $\mathcal{D}$
Ensure: Quantized weights and activation metadata
Cache the inputs to the first Transformer layer using $\mathcal{D}$.
for each Transformer layer **do**
  for each projection group **do**
    Collect representative activation rows for the group.
    if LET or LWC is used **then**
      Optimize the conditioning and clipping parameters using Eq. B.1.
    end if
    Fit activation metadata using Algorithm 2.
    Quantize the calibration activations to obtain $\widehat{X}$.
    if activation-error compensation is used **then**
      Compute the per-channel gains using Eq. B.8.
      Apply the gains to the corresponding weights.
    end if
    Construct $H = 2\widehat{X}\widehat{X}^T/N$.
    for each projection in the group **do**
      Fit its AdaPivot weight codebook using Algorithm 3.
      Quantize its weights with GPTQ sequential error propagation using the fitted codebook.
    end for
    Use the fitted activation quantizer in subsequent calibration passes.
  end for
  Fit separate activation codebooks for post-RoPE queries and keys, and for values.
  Quantize queries and keys before their matrix multiplication, and quantize values before the probability–value multiplication.
  Keep attention probabilities in their original precision.
  if attention LET is used **then**
    Optimize the equivalent Q–K and V–output scales against Transformer-layer output reconstruction.
  end if
  Run the quantized layer and cache its outputs for the next layer.
end for
return the quantized model and metadata.

B Calibration and Fitting Procedures

B.1 Layer-Wise Calibration

AdaPivot processes Transformer layers sequentially and propagates the output of each quantized layer to the next layer. For each projection group, it fits an activation codebook, constructs the GPTQ Hessian from quantized calibration activations, and quantizes the corresponding weights. The calibration workflow is summarized in Algorithm 1.

Optional conditioning. When enabled, LET applies a positive diagonal transformation $D = \mathrm{diag}(\boldsymbol{d})$ to the weights and the inverse transformation to the activations. LWC clips positive and negative weights separately for each output channel. Let Y_m denote the full-precision output of projection m, and let $\widehat{Y}_m$ denote the output using the transformed, clipped, and quantized weights and activations. The normalized reconstruction objective used to optimize these parameters is

$$\mathcal{L}_{\mathrm{rec}} = \frac{1}{P}\sum_{m=1}^{P}\frac{\|Y_m - \widehat{Y}_m\|_F^2}{\|Y_m\|_F^2 + \epsilon}. \tag{B.1}$$

After selecting the conditioning parameters, the weight and activation codebooks are refitted.

Algorithm 2 Dynamic activation-codebook fitting

Require: Calibration activations, core candidates $\mathcal{C}_x$, stride candidates $\mathcal{S}_x$, clipping candidates $\mathcal{B}$, pivot radius R_G
Ensure: Activation metadata $(C, G, S, \beta^+, \beta^-)$

```
for each activation group do
    Initialize the best candidate to have infinite loss.
    for all C ∈ C_x, S ∈ S_x do
        for h = 0, ..., R_G do
            G ← min(C + h, G_max).
            for all β ∈ B do
                Set β⁺ = β⁻ = β.
                Compute each calibration row's scale using Eq. B.3.
                Evaluate the importance-weighted reconstruction loss over calibration rows.
                Retain the lowest-loss candidate.
            end for
        end for
    end for
    Fix the selected (C, G, S) and search (β⁺, β⁻) ∈ B × B.
    Retain the lowest-loss clipping pair.
end for
return the selected metadata.
```

B.2 Dynamic Activation-Codebook Fitting

For an activation group, let m^+ and m^- be its maximum positive and negative magnitudes. Since the additional code represents the negative tail endpoint, the positive and negative endpoint indices are

$$z^+ = G + S(T-1), \qquad z^- = G + ST. \tag{B.2}$$

Given offline clipping ratios β^+ and β^-, the runtime scale is

$$\Delta_x = \max\left\{\frac{\beta^+ m^+}{z^+}, \frac{\beta^- m^-}{z^-}, \epsilon\right\}. \tag{B.3}$$

Values beyond the clipped range saturate at the corresponding endpoint.

The activation fitter selects (C, G, S) and (β^+, β^-) by minimizing calibration reconstruction error, evaluating each candidate with the same dynamic-scale rule as at inference. For activation channel i, the importance weight α_i is the sum of squared weights over the projections sharing that activation, as defined in Section 3.2. When nonuniform channel weights are used, they are normalized to have unit mean within each activation group.

The implementation first searches the discrete geometry with symmetric clipping, $\beta^+ = \beta^-$, and then refines the positive and negative clipping ratios independently while keeping the selected geometry fixed. At inference, only Δ_x is recomputed.

B.3 Weight-Codebook Fitting and GPTQ Quantization

For fixed codebook indices, let $y_i = |w_i|$ and let z_i be the assigned normalized magnitude:

$$z_i = \begin{cases} k_i, & \text{if entry } i \text{ is assigned to core index } k_i, \\ G + Sj_i, & \text{if entry } i \text{ is assigned to tail index } j_i. \end{cases}$$

The scale update is the constrained least-squares solution

$$\Delta^* = \max\left\{\epsilon, \frac{\sum_i \alpha_i z_i y_i}{\sum_i \alpha_i z_i^2}\right\}. \tag{B.4}$$

This is the positive-scale version of the conditional update in Eq. 9.

For fixed tail assignments and scale, the real-valued pivot proposal is

$$\widetilde{G} = \frac{\sum_{i \in \mathcal{I}_t} \alpha_i \left(y_i/\Delta - Sj_i\right)}{\sum_{i \in \mathcal{I}_t} \alpha_i}, \tag{B.5}$$

Algorithm 3 Integer-constrained AdaPivot weight-codebook fitting

Require: Weight group $\boldsymbol{w}$, importance weights $\boldsymbol{\alpha}$, core candidates $\mathcal{C}$, stride candidates $\mathcal{S}$, scale quantiles $\mathcal{Q}$, pivot radius R_G
Ensure: Stored codebook parameters (Δ, C, G, S)
Initialize the best candidate to have infinite loss.
for all $q \in \mathcal{Q}$, $C \in \mathcal{C}$, $S \in \mathcal{S}$ **do**
  Initialize $\Delta \leftarrow \mathrm{Quantile}_q(\{|w_i|\})/(C - \frac{1}{2})$ and $G \leftarrow C$.
  for at most R iterations **do**
    Assign entries to their nearest codebook levels.
    Update Δ using Eq. B.4.
    Compute $\widetilde{G}$ using Eq. B.5.
    Form feasible local integer pivot candidates, including $G = C$.
    for all candidate pivots G' **do**
      Reassign levels using (C, G', S, Δ) and refit Δ using Eq. B.4.
      Cast Δ to its metadata storage precision.
      Recompute assignments and evaluate the weighted reconstruction loss using the stored scale.
      Retain the lowest-loss candidate.
    end for
    if assignments have stabilized **then**
      break
    end if
  end for
  Update the best candidate across all initializations.
end for
Also evaluate the signed-uniform candidate $(C, G, S) = (M - 1, M - 1, 1)$.
return the lowest-loss stored candidate.

when the denominator is positive. If there are no positive-weight tail assignments, the fitter uses $G = C$. The integer pivot candidates include $G = C$ and the feasible integers near $\mathrm{round}(\widetilde{G})$.

The fitted codebook is held fixed during the GPTQ column sweep. Let $\widehat{X}$ be the quantized calibration activations and define

$$H = \frac{2}{N}\widehat{X}\widehat{X}^T, \qquad U^T U = (H + \lambda I)^{-1}, \qquad \alpha_i = \frac{1}{U_{ii}^2}. \tag{B.6}$$

Here, U is the upper-triangular factor used by GPTQ, and α_i is the importance weight used when fitting the weight codebook. For each column i, GPTQ computes

$$\boldsymbol{e}_i = \frac{\boldsymbol{w}_i - \widehat{\boldsymbol{w}}_i}{U_{ii}}, \qquad W_{:,i+1:} \leftarrow W_{:,i+1:} - \boldsymbol{e}_i U_{i,i+1:}. \tag{B.7}$$

Thus, AdaPivot replaces the reconstruction levels used by GPTQ while retaining its sequential error-propagation update.

When activation-error compensation is enabled, let $\boldsymbol{u}_i$ and $\boldsymbol{v}_i$ denote the calibration values of transformed activation channel i before and after quantization, respectively. The least-squares channel gain is

$$r_i = \mathrm{clip}\left(\frac{\langle \boldsymbol{u}_i, \boldsymbol{v}_i \rangle}{\max(\langle \boldsymbol{v}_i, \boldsymbol{v}_i \rangle, \epsilon)}, r_{\min}, r_{\max}\right). \tag{B.8}$$

The gain is applied to the corresponding input column of the weight matrix before weight-codebook fitting.

B.4 Attention-Operand Quantization

AdaPivot fits separate activation formats for post-RoPE queries and keys, and for values. Queries and keys are quantized before their matrix multiplication; values are quantized before the probability–value multiplication. Attention probabilities remain in their original precision. Each attention operand uses one activation group per token and head.

Table S1: Supplementary Llama2-13B weight-only comparison. PPL is lower-is-better; Avg. is the mean accuracy (%) over the five downstream tasks.

Method	Precision	WikiText-2 PPL ↓	C4 PPL ↓	Avg. ↑
FP16	W16A16	4.88	6.47	71.12%
RTN-MinMax	W4A16	5.04	6.64	70.43%
RTN-Clip	W4A16	5.14	6.72	70.32%
GPTQ	W4A16	5.02	6.59	70.11%
AWQ	W4A16	–	–	–
AdaPivot	**W4A16**	**4.97**	**6.55**	**71.25%**

When attention LET is enabled, positive channelwise scales preserve the floating-point products:

$$QK^T = \left(Q\,\mathrm{diag}(\boldsymbol{s}_{qk}^{-1})\right)\left(K\,\mathrm{diag}(\boldsymbol{s}_{qk})\right)^T, \qquad PV = \left[P\left(V\,\mathrm{diag}(\boldsymbol{s}_{vo}^{-1})\right)\right]\mathrm{diag}(\boldsymbol{s}_{vo}). \tag{B.9}$$

These scales are optimized against Transformer-layer output reconstruction while keeping the fitted discrete codebook parameters fixed.

C Supplementary Weight-Only Results

Table S1 reports a weight-only W4A16 comparison on Llama2-13B, which isolates the effect of weight quantization from activation error. AdaPivot achieves the lowest perplexity among all quantized methods on both WikiText-2 and C4, outperforming GPTQ, while its average zero-shot accuracy (71.25%) matches the FP16 baseline (71.12%). Notably, AdaPivot shares the same GPTQ-style error compensation with the GPTQ baseline, so the improvement can be attributed directly to the adaptive core–tail format rather than to the reconstruction procedure. These results indicate that the proposed format remains beneficial even in the weight-only setting, where the quantization error is dominated by the weight distribution alone.