RQT: Hierarchical Residual Quantization for Multi-Model Compression

Tianqi Chen[1,2] , **Peisong Wang**[1,2,3†], **Weixiang Xu**[1], **Zeyu Zhu**[1],
Jian Cheng[1,2,3,4†]
[1] C[2]DL, Institute of Automation, Chinese Academy of Sciences
[2]School of Artificial Intelligence, University of Chinese Academy of Sciences
[3]AIRIA [4]Maicro.ai
{chentianqi2023,chenyuanteng2024,xuweixiang2018,zhuzeyu2021}@ia.ac.cn,
{peisong.wang,jcheng}@nlpr.ia.ac.cn

Abstract

Delta compression methods focus on efficiently serving multiple uniquely fine-tuned models, each tailored to specific tasks and user requirements. These approaches decompose a fine-tuned LLM into a base model and corresponding delta weights, which are compressed using low-rank or low-bit representations to reduce storage costs. However, their effectiveness is highly sensitive to the magnitude of the model deltas—a factor directly influenced by the scale of the training data. We propose the **Residual Quantization Tree (RQT)**, a hierarchical quantization framework that automatically shares low-bit integer weights across similar fine-tuned models. The RQT construction employs a two-phase greedy algorithm: a bottom-up aggregation of models based on weight matrix similarity and top-down residual quantization, in which each node optimizes the quantization parameters and then delegates residual errors to child nodes. We evaluate RQT on fine-tuned models across mathematics, coding, chatbot, and Chinese LLMs. The results show that RQT achieves an average accuracy degradation of approximately 3% (comparable to previous 4-bit post-training quantization) while maintaining an effective bitwidth of around 2 bits.

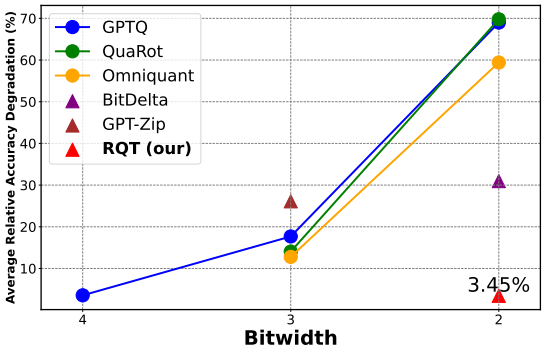


Figure 1: We evaluate RQT on fine-tuned models across mathematics, coding, chatbot, and Chinese LLMs. The results show that RQT achieves an average accuracy degradation of approximately 3% (comparable to previous 4-bit post-training quantization) while maintaining an effective bitwidth of around 2 bits.

1 Introduction

Large Language Models (LLMs), such as GPT (OpenAI, 2023b) and LLaMA (Touvron et al., 2023), have demonstrated exceptional performance and are widely applied in various applications, including chatbots (Chiang et al., 2023; OpenAI, 2023b; Touvron et al., 2023; Tu et al., 2024) and coding assistants (Luo et al., 2024; Wei et al., 2024; Rozière et al., 2023). These models achieve high accuracy in target domains through a two-stage process: pre-training on a large corpus of text data, followed by fine-tuning on task-specific datasets, such as code (Luo et al., 2024), math (Xiong et al., 2024), or human preferences (OpenAI, 2023b). Cloud AI infrastructure providers, such as OpenAI (OpenAI, 2023a), Google (Google, 2023), and Microsoft (Microsoft, 2023) offer APIs that allow users to fine-tune pre-trained LLMs with their own data, enabling the creation of customized model variants. This paradigm has driven substantial research into developing efficient methods for serving millions of uniquely fine-tuned models, each tailored to individual tasks and user requirements (Liu et al., 2024a; Ping et al., 2024; Yao and Klimovic, 2023; Ryu et al., 2023).

Although systems like Punica (Chen et al., 2024) and S-LoRA (Sheng et al., 2023) can scale to serve thousands of Low-Rank Adaptation (LoRA) adapters, these methods are incompatible with traditional full-parameter fine-tuning approaches. Notably, most models on the Open LLM Leaderboard (Fourrier et al., 2024) still rely on full-parameter fine-tuning due to the performance gap between full-parameter fine-tuning and LoRA (Biderman et al., 2024; Ding et al., 2023). To address these

† Corresponding authors.

Findings of the Association for Computational Linguistics: ACL 2025, pages 10644–10660

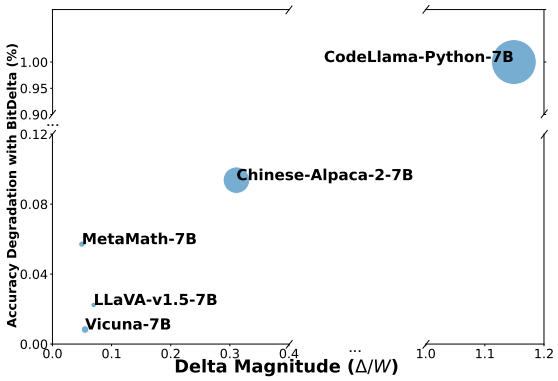


Figure 2: This figure illustrates the relationship between relative accuracy degradation and delta magnitude across models for various tasks. The circle size represents the training dataset size.

challenges, delta compression methods (Liu et al., 2024a; Ping et al., 2024; Yao and Klimovic, 2023; Ryu et al., 2023) decompose the fine-tuned model weights W_{fine} into the weights of the base pre-trained model W_{base} and the delta induced by the fine-tuning process Δ ($\Delta = W_{fine} - W_{base}$). This delta is then compressed using techniques such as quantization (Liu et al., 2024a; Isik et al., 2023), pruning (Yu et al., 2024), low-rank matrix factorization (Ping et al., 2024; Ryu et al., 2023), or combinations of these methods.

Delta compression methods are based on the observation that model deltas generally have smaller magnitudes and inherent sparsity (Yu et al., 2024; Liu et al., 2024a; Yao and Klimovic, 2023). As a result, deltas tend to be more compressible than the original model weights. For example, BitDelta (Liu et al., 2024a) achieves 1-bit quantization of delta weights, while DARE (Yu et al., 2024) can prune up to 90% of delta weights. However, this assumption primarily applies when fine-tuning on small datasets, such as instruct tuning (Chiang et al., 2023; Touvron et al., 2023; Xiong et al., 2024; Liu et al., 2023a). As shown in Figure 2, as the size of the training data increases, the relative magnitude of the delta $|\Delta|/|W_{fine}|$ also increases, leading to a greater degradation of accuracy when applying delta compression methods to Chinese and Code LLM. Consequently, BitDelta confines its experiments to chat models, such as LLaMA-2-chat (Touvron et al., 2023), Vicuna (Chiang et al., 2023), and WizardLM (Xu et al., 2024). In contrast, while Delta-CoMe (Ping et al., 2024) broadens its scope by evaluating delta compression methods across various tasks, including mathematics, coding, chat, and multi-modal applications, it still faces a limitation: it selects different base models for each task to ensure that the fine-tuned models remain sufficiently close to their base models.

This paper proposes a novel quantization framework, the **Residual Quantization Tree (RQT)**, for efficiently serving multiple fine-tuned models. An RQT is a hierarchical structure in which the nodes share low-bit integer weights between similar models. As illustrated in Figure 3, models are recursively partitioned into child nodes until reaching the leaf nodes, each corresponding to an individual model. The quantized weights are computed by aggregating the quantized values at each node along the path from the root to the corresponding leaf. To construct an RQT, we introduce a practical two-step greedy approach. (a) Bottom-up tree construction: Starting from leaf nodes (individual models), we iteratively merge node pairs with minimal weight matrix divergence (measured by L_2 distance), forming parent nodes until reaching the root. (b) Top-down residual quantization: Beginning at the root, we compute optimal low-bit integer weights for contained models at each node, then propagate residual errors to child nodes for refinement, recursively minimizing the weight reconstruction error. Our contributions are threefold:

- We propose the **Residual Quantization Tree (RQT)**, which addresses the sensitivity of previous delta compression methods to the magnitude of delta values.
- To construct an RQT, We propose an efficient two-step greedy approach that decouples model assignment and residual quantization.
- Experiments demonstrate the effectiveness of RQT in comparison with previous PTQ and delta-compression methods. Even with an average bitwidth of approximately 2 bits, RQT achieves an average accuracy degradation of approximately 3% on fine-tuned LLaMA-2-7B models across various tasks.

2 Related Works

2.1 Model Quantization

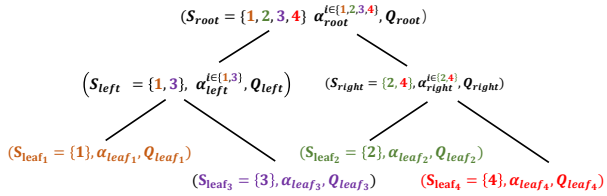
Quantization has been extensively applied to accelerate model inference (Jacob et al., 2018; Nagel et al., 2020; Li et al., 2021; Xiao et al., 2023b; Chen et al., 2023), and it has become a crucial technique for LLMs in the current era of rapid development

(Wei et al., 2022; Xiao et al., 2023a; Lin et al., 2023; Frantar et al., 2022; Liu et al., 2023b). Depending on whether activations are quantized, quantization can be categorized into weight-only quantization and weight-activation quantization. Weight-only quantization reduces memory usage and inference latency by quantizing weights into low-bit precision while keeping activations in floating-point. For example, GPTQ (Frantar et al., 2022) introduces a layer-wise quantization approach based on approximate second-order information. OmniQuant (Shao et al., 2023) incorporates learnable parameters to clip extreme weight values and shift the quantization challenge from activations to weights. Recently, QuaRot (Ashkboos et al., 2024) and SpinQuant (Liu et al., 2024b) facilitate quantization by rotating LLMs to eliminate outliers in activations without affecting the output. Despite these advancements, ultra-low-bit quantization (e.g., 2-bit) continues to experience significant performance degradation. Additionaly, QuIP (Chee et al., 2023) and AQLM (Egiazarian et al., 2024) utilize vector quantization to better maintaining the shape of the source distribution, especially under lower bitwidth. However, these PTQ methods still suffer non-negligible performance degradation at 2-bit quantization.

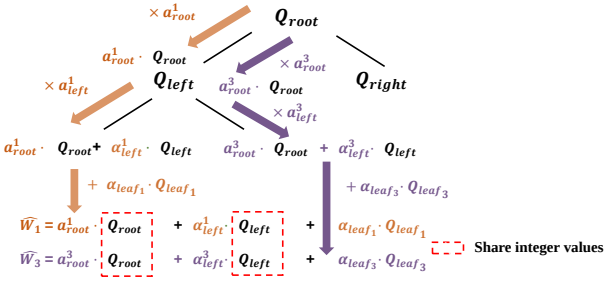
2.2 Delta Compression

The rise of millions of uniquely fine-tuned models has recently driven significant research interest in delta compression methods. Specifically, delta compression involves subtracting the base pre-trained model weights from fine-tuned weights to obtain the model delta. Since model deltas generally exhibit smaller magnitudes and inherent sparsity, they are often more compressible. For example, GPT-Zip (Isik et al., 2023) quantizes model deltas into 2/3/4 bits using the post-training quantization (PTQ) method GPTQ (Frantar et al., 2022). DARE (Yu et al., 2024) eliminates the majority of delta weights (up to 90%) with minimal impact on the performance of aligned LLMs. BitDelta (Liu et al., 2024a) binarizes model deltas and fine-tunes the quantization scales through knowledge distillation. Delta-CoMe (Ping et al., 2024) applies singular value decomposition (SVD) to deltas and assigns varying bitwidths to different singular vectors based on their singular values.

Although these methods achieve high compression ratios by quantizing deltas, they face limitations when applied to models across diverse tasks.



(a) Residual Quantization Tree



(b) Computation of quantized weights

Figure 3: An illustration of an RQT with four fine-tuned models. Different colors denote each model. The quantized weights $\widehat{W}_1$ and $\widehat{W}_3$ are computed by summing along the path from the root to the leaf. They use the shared integer values Q_{root} and Q_{left} in their common ancestor node.

In other words, the assumption that model deltas are smaller in magnitude than the original weights is not always valid, especially when fine-tuning with large datasets, such as those for code and Chinese language models.

3 Methods

3.1 Formulation

Let $\{W_i \in \mathbb{R}^{d_o \times d_i}\}_{i=1}^N$ represent the weight matrices of a target linear layer across N fine-tuned models, where d_i and d_o represent the input and output dimensions, respectively. We define the Residual Quantization Tree (RQT) as follows:

Definition 1 (Residual Quantization Tree)**.** Given a set of weight matrices $\{W_i \in \mathbb{R}^{d_o \times d_i}\}_{i=1}^N$ from N fine-tuned models, a Residual Quantization Tree is a tree where each node v contains:

- $S_v \subseteq \{1, \ldots, N\}$: The index set of assigned models to node v.
- $Q_v \in \mathbb{Z}^{d_o \times d_i}$: A *shared* integer matrix for quantizing *all* weight matrices assigned to node v, i.e., $\{W_i \mid i \in S_v\}$;
- $\Sigma_v = \{\alpha_v^i \in \mathbb{R}^{d_o} \mid i \in S_v\}$: The *model-specific* quantization scales, where each α_v^i is specific to W_i at node v;
- $C_v = \{v_1, v_2, \ldots, v_m\}$: The child nodes of v satisfying $\bigcup_{j=1}^m \mathcal{S}_{v_j} = \mathcal{S}_v$ with $\mathcal{S}_{v_j} \cap \mathcal{S}_{v_k} = \emptyset$

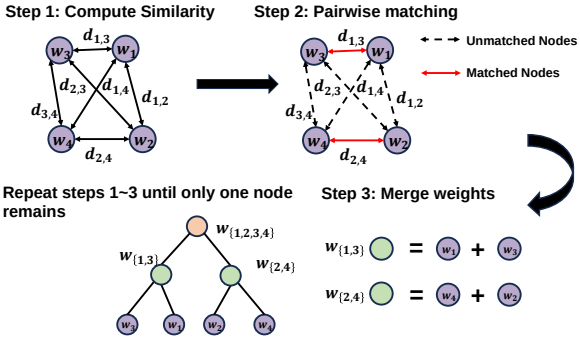


Figure 4: Illustration of the bottom-up construction process of the RQT.

The root node encompasses all models with $S_{\text{root}} = \{1, \ldots, N\}$. Every non-leaf node v distributes its assigned models S_v to child nodes C_v until the leaf nodes are reached, each containing a single model ($|S_{\text{leaf}}| = 1$). At each node v, weight matrix W_i is quantized as $\alpha_v^i \cdot Q_v$, using the shared low-bit values Q_v and the individual per-channel quantization scales α_v^i. This architecture achieves higher effective bitwidth than the average storage bitwidth, establishing superior compression-performance trade-offs compared to conventional quantization methods.

Below, we describe how to compute the quantized weights given the RQT:

Definition 2. For weight matrix W_i, its quantized weights $\widehat{W_i}$ are computed through summation along the path from the root to the corresponding leaf:

$$\widehat{W_i} = \sum_{v \in \mathcal{P}(i)} \alpha_v^i \cdot Q_v \tag{1}$$

where $\mathcal{P}(i)$ denotes the path from root to leaf node contains i-th model ($S_{\text{leaf}} = \{i\}$).

Remarks. RQT fundamentally advances delta compression through its hierarchical residual decomposition. Unlike conventional approaches that decompose model weights as $\Delta = W_{fine} - W_{base}$ with manually selected W_{base}, the RQT has two advantages: (a) The hierarchical representation of inter-model similarities aligns naturally with multi-stage LLM fine-tuning pipelines. (b) RQT achieves adaptive bitwidth allocation compared to delta compression, making it less sensitive to delta magnitude and more robust to task diversity.

3.2 Algorithm

In this section, we propose a greedy two-step approach for constructing an RQT. For simplicity, we assume that the tree is binary ($|C_v| = 2$ for all nodes) and that the number of models N is a power of 2. In terms of bit allocation, we apply binary quantization (1-bit) to all nodes except the root node, which is allocated higher bits (e.g., 4-bits). This simplification yields a complete binary tree with an average bitwidth of $\frac{4+(2N-2)\times 1}{N} \approx 2$ bits. Section 4.4 later relaxes these assumptions to handle arbitrary N by incorporating new models into an existing RQT.

3.2.1 Step 1: Tree Construction

Intuitively, models with higher similarity should be positioned closer in the RQT to enable them to have more common ancestor nodes. For example, in Figure 3, the weight matrices W_1 and W_3 should originate from similar models (e.g., both are math LLMs), as they utilize the same low-bit matrix Q_{left} in the second layer. The same applies to W_2 and W_4.

To achieve this, we propose a greedy bottom-up tree construction method based on the similarity of weight matrices. An illustration of this procedure is provided in Figure 4. Specifically, we start with N leaf nodes $\{v_i \mid i \in 1, \ldots, N\}$, each containing a single model $S_{v_i} = \{i\}$. Next, we pair nodes by minimizing the total distance between their respective weight matrices:

$$\begin{aligned} &\min_{\{x_{ij}\}_{1\le i<j\le N}} \sum_{1\le i<j\le N} x_{ij} \cdot d_{ij}, \\ &\text{subject to:} \quad \sum_{j\neq i} x_{ij} = 1, \quad x_{ij} \in \{0,1\}. \end{aligned} \tag{2}$$

Here, x_{ij} is a binary variable indicating whether nodes i and j are paired, and d_{ij} represents the distance between the weight matrices W_i and W_j, chosen as the L_2 distance in this work:

$$d_{ij} = \|W_i - W_j\|_2. \tag{3}$$

This problem can be solved using dynamic programming or linear integer programming techniques, as detailed in (Cormen et al., 2022). Once the optimal pairs $\{x_{ij}\}_{1\le i<j\le n}$ are determined, we merge the matched nodes into their parent nodes by averaging their weights:

$$\begin{aligned} S_{\text{parent}(v_i,v_j)} &= S_{v_i} \cup S_{v_j}, \\ C_{\text{parent}(v_i,v_j)} &= \{v_i, v_j\}, \\ W_{\text{parent}(v_i,v_j)} &= \frac{1}{2}(W_i + W_j). \end{aligned} \tag{4}$$

This iterative process—which comprises distance computation, matching, and merging—continues until a single root node remains.

Algorithm 1 Tree_Construction

Input: A set of N weight matrices $\Theta = \{W_i \mid i = 1, \ldots, N\}$
Output: An incomplete residual quantization tree T with defined S and C
Let $\Phi = \{v_i \mid i = 1, \ldots, N\}$ be the set of leaf nodes.
for $i = 1$ to N **do**
    Initialize $S[v_i] = \{i\}$, $C[v_i] = \emptyset$
end for
while $|\Phi| > 1$ **do**
    Compute x_{ij} according to Equation 2
    for each pair of nodes $(v_i, v_j) \in \Phi \times \Phi$ **do**
        if $x_{ij} = 1$ **then**
            Create a new parent node v_{new}
            $W_{\text{new}} \leftarrow \frac{1}{2}(W_i + W_j)$
            $S[v_{\text{new}}] \leftarrow S[v_i] \cup S[v_j]$
            $C[v_{\text{new}}] \leftarrow \{v_i, v_j\}$
            Add v_{new} to Φ and W_{new} to Θ
            Remove v_i and v_j from Φ
            Remove W_i and W_j from Θ
        end if
    end for
end while
return the root node in Φ

The complete pseudocode is formally described in Algorithm 1. Crucially, the newly generated merged weights are exclusively employed for similarity computation during tree construction. Upon completing the initial phase, we obtain an incomplete RQT with established model indices S_v and child nodes mapping C_v. The subsequent phase involves computing quantization parameters Q_v and Σ_v for each node.

3.2.2 Step 2: Residual Quantization.

In the second step, we apply residual quantization in a top-down manner. Specifically, we compute the optimal shared binary weights Q and quantization scales $\Sigma = \{\alpha_i | i \in S\}$ by minimizing the MSE for each node. Without loss of generality we assume W and Q are vectors in $\mathbb{R}^n$, where $n = d_i$ in per-channel quantization:

$$\begin{aligned} J(Q, \Sigma) &= \sum_{i \in S} ||W_i - Q \cdot \alpha_i||_2^2 \\ &= \sum_i \alpha_i^2 Q^T Q - 2\alpha_i Q^T W_i + W_i^T W_i \end{aligned} \tag{5}$$

This objective differs from those of previous binary neural networks (Rastegari et al., 2016; Courbariaux and Bengio, 2016) in that the binary weights are shared across multiple models. To address this, we propose an iterative method to compute the optimal values for α_i^* and Q^* using Equation 6 and Equation 7, respectively. Specifically, the optimal solution for α_i is obtained by setting

Algorithm 2 Residual_Quantization

Input: A set of weights $\Theta = \{W_i \mid i = 1, \ldots, N\}$
Input: An incomplete residual quantization tree T with only defined C and S
Output: A complete residual quantization tree T with defined Q and Σ
while not converged **do**
    Compute α_i using Equation 6
    Compute Q using Equation 7
end while
Update $W_i \leftarrow W_i - Q \cdot \alpha_i$ for all i
for each $T_j \in C[T]$ **do**
    Residual_Quantization($\{W_i \mid i \in S[T_j]\}, T_j$)
end for
return T

the partial derivative of the objective function with respect to α_i to zero:

$$\alpha_i^* = \frac{Q^T W_i}{Q^T Q} \tag{6}$$

Similarly, when fixed scales α_i, the optimal solution for Q can be derived by taking the sign of the weighted summation of all weights:

$$Q^* = \text{sign}\left(\sum_i \alpha_i W_i\right) \tag{7}$$

Equations 6 and 7 are applied alternately until convergence is reached. Next, the residual weights are computed and passed to the child nodes. The complete procedure is formally described in Algorithm 2.

$$W_i \leftarrow W_i - \alpha_i \cdot Q \tag{8}$$

4 Experiments

4.1 Experiment Setup

Tasks and Models. We conduct comprehensive evaluations across **four representative application domains** of aligned LLMs: **Mathematical Reasoning**, **Code Generation**, **Chatbots**, and **Chinese Language Understanding**. For each domain, we systematically select **four open-source models** fine-tuned on the LLaMA-2-7B model, creating a 16-model benchmark to establish a five-level RQT.

- **Mathematical Reasoning**: Employing GSM8K (Cobbe et al., 2021) and MATH (Hendrycks et al., 2021) benchmarks, we assess MetaMath-7B-V1.0 (Xiong et al., 2024), WizardMath-V1.0 (Luo et al., 2023), Xwin-Math-7B-V1.1 (Li et al., 2024), and MuggleMath-7B (Li et al., 2023a).

- **Code Generation**: Assessing Pass@1 (%) performance on HumanEval (Chen et al., 2021) and MBPP (Austin et al., 2021), we assess CodeLLaMA-7B-Python (Rozière et al., 2023), WizardCoder-Python-7B-V1.0 (Luo et al., 2024), Magicoder-S-CL-7B (Wei et al., 2024), and ReflectionCoder-CL-7B (Ren et al., 2024).

- **Chatbots**: Measuring accuracy on ARC-Easy (Clark et al., 2018) and HellaSwag (Zellers et al., 2019), we assess LLaMA-2-7B (Touvron et al., 2023), LLaMA-2-chat-7B (Touvron et al., 2023), Vicuna-v1.5 (Chiang et al., 2023), and Vicuna-7B-v1.5-16k (Chiang et al., 2023).

- **Chinese Language Processing**: Utilizing C-Eval (Huang et al., 2023) and CMMLU (Li et al., 2023b), we assess Chinese-Alpaca-2-7B (Cui et al., 2023), Chinese-LLaMA-2-7B (Cui et al., 2023), and their 16k variants.

Setup. We employ per-channel binary quantization (1-bit) for non-root nodes with 4-bit precision reserved for the root node, achieving an effective bitwidth of $\frac{4+30}{16} = 2.125$. Building upon BitDelta (Liu et al., 2024a), we optimize quantization scales α_i through end-to-end training while keeping the low-bit model weights frozen. This approach effectively incorporates activation distribution characteristics while significantly reducing GPU memory consumption through weight freezing. The calibration process utilizes 800 randomly sampled 128-token sequences from the C4 corpus (Pal et al., 2023). We perform 3 training epochs using the AdamW optimizer with a learning rate of 5×10^{-6}, executed on a single NVIDIA A800 GPU (80GB). The complete distillation process requires approximately 20 minutes for the 7B parameter model. Furthermore, inspired by QuaRot (Ashkboos et al., 2024) and SpinQuant (Liu et al., 2024b), we implement pre-quantization orthogonal rotation on weight matrices to enhance quantization robustness while introducing no additional inference overhead.

Baselines. We evaluate our framework against two categories of compression approaches: (1) PTQ methods including GPTQ (Frantar et al., 2022), OmniQuant (Shao et al., 2023), and QuaRot (Ashkboos et al., 2024); and (2) delta compression techniques represented by GPT-Zip (Isik et al., 2023) and BitDelta (Liu et al., 2024a).

- For PTQ baselines, we employ group-wise quantization with a group size of 128, achieving an average storage overhead of 0.125 bits per element.

- For delta compression methods, we implement BitDelta and GPT-Zip using the LLaMA-2-7B (16-bit floating-point) as the base model, quantizing 16 model deltas to 1-bit and 2-bit precision, respectively. This configuration yields effectively 2 bits (BitDelta: $\frac{16\times1+16}{16}$) and 3 bits (GPT-Zip: $\frac{16\times2+16}{16}$) per element when accounting for base model storage. For single-task settings, we use LLaMA-2 as the base model for the math and chat tasks, and Chinese-LLaMA-2-7B or CodeLLaMA-Python for the Chinese and code tasks. For multi-task settings, we use LLaMA-2 as the base model for all tasks.

4.2 Main Results

Comparison with PTQ Methods. Table 1 presents aggregated results across two evaluation datasets for LlaMA2-7B fine-tuned variants (more detailed results are provided in the Appendix). Our framework employs per-channel binary quantization (1-bit) for non-root nodes with 4-bit precision reserved for the root node, achieving an effective bitwidth of $\frac{4+30}{16} = 2.125$ bits/parameter – equivalent to conventional 2-bit group-wise quantization with group size 128 in storage cost. Remarkably, despite this comparable bandwidth, our method surpasses existing 3-bit PTQ approaches in accuracy, particularly on complex multi-step reasoning tasks (mathematical problem-solving and code generation). For instance, in mathematical reasoning evaluation, standard PTQ methods exhibit catastrophic degradation with accuracy dropping to nearly zero, while our approach maintains robust performance with a maximum degradation of only 2.51%.

Comparison with Delta Compression Methods. Figure 5 and Table 1 demonstrate the superiority of RQT over delta compression baselines (BitDelta (Liu et al., 2024a), GPT-Zip (Isik et al., 2023)) when models span multiple tasks. For single-task settings, we adopt LLaMA-2 as the base model for the math and chat tasks, and use Chinese-LLaMA-2-7B and CodeLLaMA-Python as the base models for the Chinese and code tasks, respectively. In multi-task settings (i.e., when the number of tasks is greater than or equal to two), we uniformly use LLaMA-2 as the base model for all

Table 1: Results of RQT for 16 models across four tasks, averaged over two evaluation datasets. Our method employs per-channel binary quantization for all nodes, except the 4-bit root node. This yields an average bit width of $(4 + 30)/16 = 2.125$. Previous PTQ methods, including GPTQ, QuaRot, and OmniQuant, utilize group-wise quantization with a group size of 128. For delta compression, we implement BitDelta and GPT-Zip, using the LLaMA-2-7B (16-bit floating-point) as the base model and quantizing 16 model deltas to 1-bit and 2-bit precision, respectively.

Method	**#Bits**	**Math**				**Code**			
		WizardMath	MetaMath	Xwin-Math	MuggleMath	CodeLLaMA-PY	Wizardcoder	Magicoder-S-CL	ReflectionCoder
FP	16	32.75	43.76	64.48	46.00	42.32	56.67	69.55	73.05
GPTQ	4.125	30.65	43.35	62.54	43.90	38.68	51.59	70.40	70.32
	3.125	21.00	38.85	56.02	34.35	23.48	38.61	55.55	57.54
	2.125	2.90	1.37	0.40	2.15	0.00	0.00	0.00	0.00
QuaRot	3.125	23.85	39.25	56.50	37.90	32.14	42.96	61.00	63.73
	2.125	0.90	1.85	1.25	1.50	0.00	0.00	0.00	0.00
OminiQuant	3.125	22.65	40.70	57.84	36.85	30.84	46.28	62.70	62.98
	2.125	1.95	8.75	2.86	4.95	4.15	1.61	17.70	18.73
BitDelta	2	30.95	41.05	57.57	42.65	0.00	0.00	0.00	0.00
GPT-Zip	3	32.10	43.40	63.21	45.70	0.00	0.00	0.00	0.00
Delta-CoME	2	31.45	42.5	59.70	43.85	0.00	0.00	0.00	0.00
RQT	**2.125**	**35.45**	**43.35**	**61.97**	**48.45**	**31.57**	**51.19**	**64.05**	**62.43**

		Chat				**Chinese**			
		LLaMA-2-Chat	Vicuna-v1.5	Vicuna-v1.5-16k	LLaMA-2	CN-Alpaca-2	CN-LLaMA-2	CN-Alpaca-2-16k	CN-LLaMA-2-16k
FP	16	74.69	74.72	75.25	76.18	40.97	28.06	37.54	27.49
GPTQ	4.125	73.73	74.27	74.67	75.73	40.05	26.91	36.56	25.84
	3.125	69.37	70.95	69.60	72.19	34.42	26.28	34.53	24.30
	2.125	26.26	32.40	28.62	31.37	24.43	24.21	25.73	26.84
QuaRot	3.125	69.99	71.50	66.81	69.65	32.91	26.22	34.52	25.97
	2.125	29.06	33.99	28.82	31.02	25.25	24.90	25.11	24.04
OminiQuant	3.125	70.50	71.47	70.92	73.31	33.79	26.33	33.73	26.31
	2.125	43.11	50.45	37.65	52.18	24.33	25.27	24.72	24.18
BitDelta	2	74.70	75.10	63.20	76.18	36.91	26.47	29.13	25.31
GPT-Zip	3	75.06	75.15	75.62	76.18	39.83	26.92	35.56	26.13
Delta-CoME	2	75.01	75.28	75.34	76.18	39.19	27.3	36.96	26.25
RQT	**2.125**	**74.92**	**75.40**	**74.79**	**75.28**	**40.38**	**27.63**	**36.07**	**27.20**

Methods	**Math**	**Code**	**Chat**	**Chinese**
Our two-step method	42.99	50.74	74.87	32.91
Top-down single-step method	40.29	49.19	74.77	30.41
Our two-step method	42.99	50.74	74.87	32.91
– w/o model-specific scales	43.32	30.41	74.46	31.87
+ with scale distillation	**47.30**	**52.58**	**75.09**	**32.76**

Table 2: Design choices of building an RQT.

Metric	**Math**	**Code**	**Chat**	**Chinese**
L_2 norm	42.99	50.74	74.87	**32.91**
Cosine Similarity	**43.56**	50.86	**74.95**	31.78
Sign Difference	42.59	**51.05**	74.80	31.78

Table 3: Comparison of different distance metrics in tree construction.

tasks. While delta compression methods achieve performance comparable to RQT in single-task scenarios with carefully selected base models, their performance deteriorates dramatically—often collapsing to near-zero—on code-related LLMs as task diversity increases.

As analyzed in Section 3.2, this collapse is due to the limited representational capacity of extremely low-bit quantization when applied to large-magnitude model deltas. In contrast, RQT avoids explicitly representing model deltas Δ by automatically assigning shared binary values across similar models, making it inherently more robust to task diversity.

4.3 Ablation Studies

Effectiveness of each component. In this section, we explore a single-step method for tree construction in a top-down manner, as an alternative to the two-step greedy approach described in Section 3.2. Specifically, at each node, we first compute the optimal quantization parameters using Equation (7) and Equation (6), and then determine how to distribute the residual weights among the child nodes using Equation 9. More details are provided in the

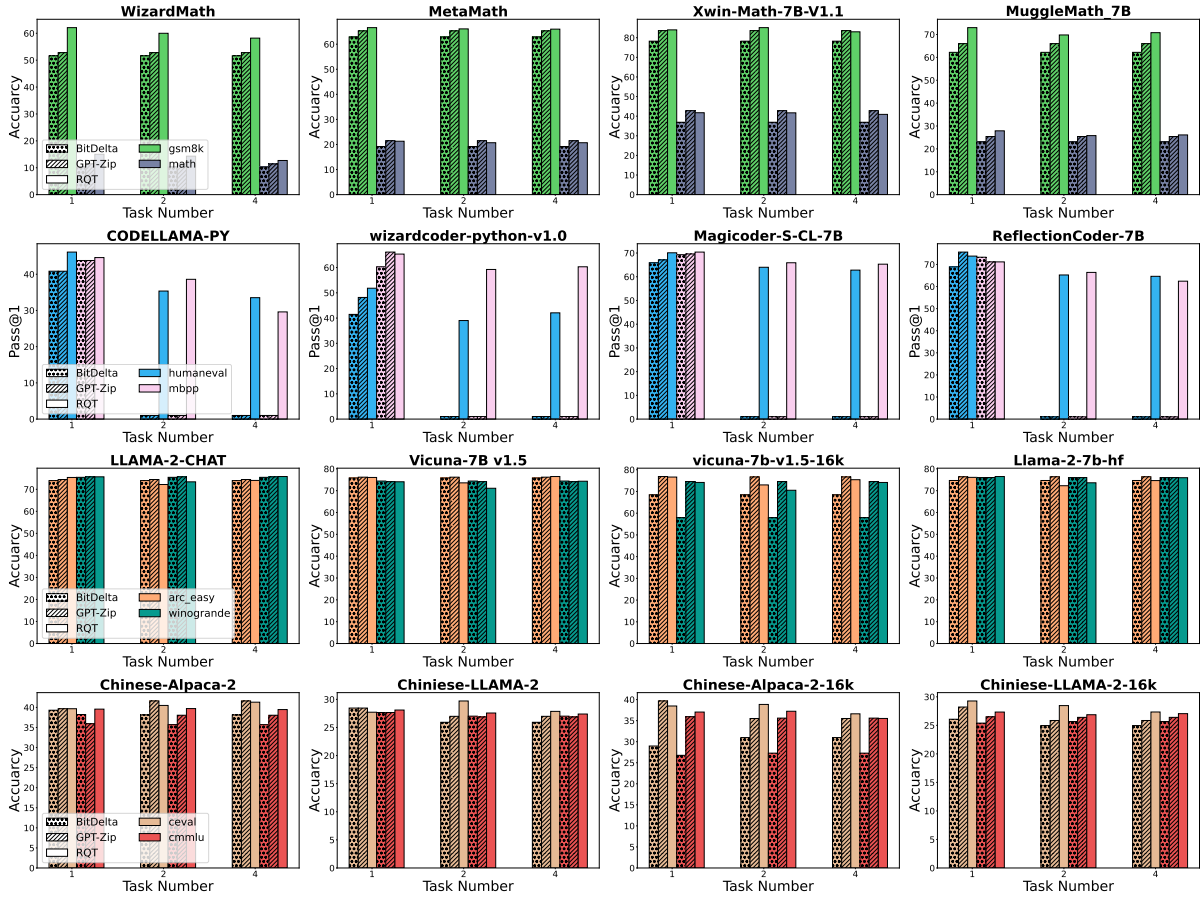


Figure 5: Results of RQT on different task numbers, where each task contains four models. Each color represents an evaluation dataset, and different hatch patterns indicate the various methods used. While delta compression methods achieve performance comparable to RQT when all models belong to a single task (the first six bars in each figure), their performance degrades significantly (to near-zero levels) on code LLMs when applied across multiple tasks. In contrast, RQT is less sensitive to delta magnitude and demonstrates greater robustness to task diversity.

appendix.

Models	Dataset	w2g128	w3g128	RQT
MAmmoTH-7B	MATH	1.38	21.94	**25.44**
LlaVA-v1.5	TextVQA	0.44	53.74	**56.64**
Magicoder-CL	MBPP	0.00	58.5.0	**64.00**
XwinLM-v0.2	HellaSwag	29.63	72.66	**76.17**

Table 4: Results of newly added models. **w2g128** and **w3g128** denote 2-bit and 3-bit quantized models, respectively, using GPTQ with a group size of 128.

Table 2 shows that the single-step method performs worse than the two-step approach introduced in Section 3.2. We attribute this to the fact that even small quantization errors at the current node can lead to incorrect model assignments, thereby reducing the robustness of the algorithm. In addition, Table 2 confirms the effectiveness of using model-specific scales and applying scale distillation.

Weight similarity metric. Table 3 examines the effect of different distance metrics on tree construction. Given the similar performance across various choices, we ultimately adopt the L_2 distance, as it aligns with the objective of residual quantization. For simplicity, scale distillation is not applied in these ablation studies.

4.4 Incorporating New Models into the RQT

In this section, we describe how to incorporate new models into an existing RQT without reconstructing the entire tree. This method relaxes the constraint that the number of models N must be a power of two and enables dynamic RQT adjustment. Specifically, we begin by identifying the optimal position for the new leaf and then perform residual quantization from the root to that leaf. During this process, we compute the optimal scales only for the new models, without modifying the shared low-bit values. The complete

pseudocode for this procedure is provided in the appendix. We evaluate this method by adding new models—MAmmoTH-7B (Yue et al., 2024), Llava-1.5 (Liu et al., 2023a) (evaluated on the TextVQA dataset (Singh et al., 2019)), Magicoder-CL (Wei et al., 2024), and Xwin-LM-v0.2 (Cui et al., 2023)—to an RQT originally built on the 16 models introduced in Section 4.1. The results are presented in Table **??**, and comparisons are made with GPTQ-based methods described in Section **??**. We leave the exploration of dynamic RQT adjustments and more general construction strategies for future work.

5 Conclusion

In this paper, we introduce the Residual Quantization Tree (RQT), which mitigates the sensitivity of previous delta compression methods to the magnitude of delta values. We also present an efficient two-step greedy approach that separates model assignment from residual quantization. Experimental results demonstrate the effectiveness of the RQT when compared to traditional PTQ methods and delta-compression methods. Even with an average bitwidth of approximately 2 bits, RQT achieves only a 3% average accuracy degradation on fine-tuned LLaMA-2-7B models across a range of tasks.

Limitations

In this paper, we introduce the Residual Quantization Tree (RQT), a hierarchical quantization framework that automatically shares low-bit integer weights across similar fine-tuned models. While the RQT can, in principle, adopt various tree structures, the algorithm proposed in Section 3.2 is limited to a complete binary tree. Further research is needed to explore dynamic adjustments to the RQT and to develop a more general construction method. Additionally, although the RQT is primarily designed for multi-model compression, it also holds potential for applications in other contexts, such as cross-layer quantization.

Acknowledgments

This work was supported in part by the National Key R&D Program of China (No.2022ZD0160304), the Natural Science Foundation of Jiangsu Province (No.BK20243051), the Science and Technology Major Special Program of Jiangsu (No.BG2024028), and the Jiangsu Key Research and Development Plan (No.BE2023016).

References

Saleh Ashkboos, Amirkeivan Mohtashami, Maximilian L. Croci, Bo Li, Martin Jaggi, Dan Alistarh, Torsten Hoefler, and James Hensman. 2024. Quarot: Outlier-free 4-bit inference in rotated llms. *CoRR*, abs/2404.00456.

Jacob Austin, Augustus Odena, Maxwell I. Nye, Maarten Bosma, Henryk Michalewski, David Dohan, Ellen Jiang, Carrie J. Cai, Michael Terry, Quoc V. Le, and Charles Sutton. 2021. Program synthesis with large language models. *CoRR*, abs/2108.07732.

Dan Biderman, Jose Javier Gonzalez Ortiz, Jacob P. Portes, Mansheej Paul, Philip Greengard, Connor Jennings, Daniel King, Sam Havens, Vitaliy Chiley, Jonathan Frankle, Cody Blakeney, and John P. Cunningham. 2024. Lora learns less and forgets less. *CoRR*, abs/2405.09673.

Jerry Chee, Yaohui Cai, Volodymyr Kuleshov, and Christopher De Sa. 2023. Quip: 2-bit quantization of large language models with guarantees. *CoRR*, abs/2307.13304.

Lequn Chen, Zihao Ye, Yongji Wu, Danyang Zhuo, Luis Ceze, and Arvind Krishnamurthy. 2024. Punica: Multi-tenant lora serving. In *Proceedings of the Seventh Annual Conference on Machine Learning and Systems, MLSys 2024, Santa Clara, CA, USA, May 13-16, 2024*. mlsys.org.

Mark Chen, Jerry Tworek, Heewoo Jun, Qiming Yuan, Henrique Pondé de Oliveira Pinto, Jared Kaplan, Harri Edwards, Yuri Burda, Nicholas Joseph, Greg Brockman, Alex Ray, Raul Puri, Gretchen Krueger, Michael Petrov, Heidy Khlaaf, Girish Sastry, Pamela Mishkin, Brooke Chan, Scott Gray, Nick Ryder, Mikhail Pavlov, Alethea Power, Lukasz Kaiser, Mohammad Bavarian, Clemens Winter, Philippe Tillet, Felipe Petroski Such, Dave Cummings, Matthias Plappert, Fotios Chantzis, Elizabeth Barnes, Ariel Herbert-Voss, William Hebgen Guss, Alex Nichol, Alex Paino, Nikolas Tezak, Jie Tang, Igor Babuschkin, Suchir Balaji, Shantanu Jain, William Saunders, Christopher Hesse, Andrew N. Carr, Jan Leike, Joshua Achiam, Vedant Misra, Evan Morikawa, Alec Radford, Matthew Knight, Miles Brundage, Mira Murati, Katie Mayer, Peter Welinder, Bob McGrew, Dario Amodei, Sam McCandlish, Ilya Sutskever, and Wojciech Zaremba. 2021. Evaluating large language models trained on code. *CoRR*, abs/2107.03374.

Tianqi Chen, Weixiang Xu, Weihan Chen, Peisong Wang, and Jian Cheng. 2023. Towards efficient and accurate winograd convolution via full quantization. In *Proceedings of the 37th International Conference on Neural Information Processing Systems*, NIPS '23, Red Hook, NY, USA. Curran Associates Inc.

Wei-Lin Chiang, Zhuohan Li, Zi Lin, Ying Sheng, Zhanghao Wu, Hao Zhang, Lianmin Zheng, Siyuan Zhuang, Yonghao Zhuang, Joseph E. Gonzalez, Ion

Stoica, and Eric P. Xing. 2023. Vicuna: An open-source chatbot impressing gpt-4 with 90%* chatgpt quality.

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. *CoRR*, abs/1803.05457.

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

Thomas H Cormen, Charles E Leiserson, Ronald L Rivest, and Clifford Stein. 2022. *Introduction to algorithms*. MIT press.

Matthieu Courbariaux and Yoshua Bengio. 2016. Binarynet: Training deep neural networks with weights and activations constrained to +1 or -1. *CoRR*, abs/1602.02830.

Yiming Cui, Ziqing Yang, and Xin Yao. 2023. Efficient and effective text encoding for chinese llama and alpaca. *arXiv preprint arXiv:2304.08177*.

Ning Ding, Yujia Qin, Guang Yang, Fuchao Wei, Zonghan Yang, Yusheng Su, Shengding Hu, Yulin Chen, Chi-Min Chan, Weize Chen, Jing Yi, Weilin Zhao, Xiaozhi Wang, Zhiyuan Liu, Hai-Tao Zheng, Jianfei Chen, Yang Liu, Jie Tang, Juanzi Li, and Maosong Sun. 2023. Parameter-efficient fine-tuning of large-scale pre-trained language models. *Nat. Mac. Intell.*, 5(3):220–235.

Vage Egiazarian, Andrei Panferov, Denis Kuznedelev, Elias Frantar, Artem Babenko, and Dan Alistarh. 2024. Extreme compression of large language models via additive quantization. In *Forty-first International Conference on Machine Learning, ICML 2024, Vienna, Austria, July 21-27, 2024*. OpenReview.net.

Clémentine Fourrier, Nathan Habib, Alina Lozovskaya, Konrad Szafer, and Thomas Wolf. 2024. Open llm leaderboard v2. `https://huggingface.co/spaces/open-llm-leaderboard/open_llm_leaderboard`.

Elias Frantar, Saleh Ashkboos, Torsten Hoefler, and Dan Alistarh. 2022. GPTQ: accurate post-training quantization for generative pre-trained transformers. *CoRR*, abs/2210.17323.

Google. 2023. Vertex AI.

Dan Hendrycks, Collin Burns, Saurav Kadavath, Akul Arora, Steven Basart, Eric Tang, Dawn Song, and Jacob Steinhardt. 2021. Measuring mathematical problem solving with the MATH dataset. In *Proceedings of the Neural Information Processing Systems Track on Datasets and Benchmarks 1, NeurIPS Datasets and Benchmarks 2021, December 2021, virtual*.

Yuzhen Huang, Yuzhuo Bai, Zhihao Zhu, Junlei Zhang, Jinghan Zhang, Tangjun Su, Junteng Liu, Chuancheng Lv, Yikai Zhang, Jiayi Lei, Yao Fu, Maosong Sun, and Junxian He. 2023. C-eval: A multi-level multi-discipline chinese evaluation suite for foundation models. *arXiv preprint arXiv:2305.08322*.

Berivan Isik, Hermann Kumbong, Wanyi Ning, Xiaozhe Yao, Sanmi Koyejo, and Ce Zhang. 2023. Gpt-zip: Deep compression of finetuned large language models. In *Workshop on Efficient Systems for Foundation Models@ ICML2023*.

Benoit Jacob, Skirmantas Kligys, Bo Chen, Menglong Zhu, Matthew Tang, Andrew G. Howard, Hartwig Adam, and Dmitry Kalenichenko. 2018. Quantization and training of neural networks for efficient integer-arithmetic-only inference. In *2018 IEEE Conference on Computer Vision and Pattern Recognition, CVPR 2018, Salt Lake City, UT, USA, June 18-22, 2018*, pages 2704–2713. Computer Vision Foundation / IEEE Computer Society.

Woosuk Kwon, Zhuohan Li, Siyuan Zhuang, Ying Sheng, Lianmin Zheng, Cody Hao Yu, Joseph E. Gonzalez, Hao Zhang, and Ion Stoica. 2023. Efficient memory management for large language model serving with pagedattention. In *Proceedings of the ACM SIGOPS 29th Symposium on Operating Systems Principles*.

Chen Li, Weiqi Wang, Jingcheng Hu, Yixuan Wei, Nanning Zheng, Han Hu, Zheng Zhang, and Houwen Peng. 2024. Common 7b language models already possess strong math capabilities. *CoRR*, abs/2403.04706.

Chengpeng Li, Zheng Yuan, Hongyi Yuan, Guanting Dong, Keming Lu, Jiancan Wu, Chuanqi Tan, Xiang Wang, and Chang Zhou. 2023a. Query and response augmentation cannot help out-of-domain math reasoning generalization. *CoRR*, abs/2310.05506.

Haonan Li, Yixuan Zhang, Fajri Koto, Yifei Yang, Hai Zhao, Yeyun Gong, Nan Duan, and Timothy Baldwin. 2023b. Cmmlu: Measuring massive multitask language understanding in chinese. *Preprint*, arXiv:2306.09212.

Yuhang Li, Ruihao Gong, Xu Tan, Yang Yang, Peng Hu, Qi Zhang, Fengwei Yu, Wei Wang, and Shi Gu. 2021. BRECQ: pushing the limit of post-training quantization by block reconstruction. In *9th International Conference on Learning Representations, ICLR 2021, Virtual Event, Austria, May 3-7, 2021*. OpenReview.net.

Ji Lin, Jiaming Tang, Haotian Tang, Shang Yang, Xingyu Dang, and Song Han. 2023. AWQ: activation-aware weight quantization for LLM compression and acceleration. *CoRR*, abs/2306.00978.

Haotian Liu, Chunyuan Li, Qingyang Wu, and Yong Jae Lee. 2023a. Visual instruction tuning.

James Liu, Guangxuan Xiao, Kai Li, Jason D. Lee, Song Han, Tri Dao, and Tianle Cai. 2024a. Bitdelta: Your fine-tune may only be worth one bit. *CoRR*, abs/2402.10193.

Zechun Liu, Barlas Oguz, Changsheng Zhao, Ernie Chang, Pierre Stock, Yashar Mehdad, Yangyang Shi, Raghuraman Krishnamoorthi, and Vikas Chandra. 2023b. LLM-QAT: data-free quantization aware training for large language models. *CoRR*, abs/2305.17888.

Zechun Liu, Changsheng Zhao, Igor Fedorov, Bilge Soran, Dhruv Choudhary, Raghuraman Krishnamoorthi, Vikas Chandra, Yuandong Tian, and Tijmen Blankevoort. 2024b. Spinquant: LLM quantization with learned rotations. *CoRR*, abs/2405.16406.

Haipeng Luo, Qingfeng Sun, Can Xu, Pu Zhao, Jianguang Lou, Chongyang Tao, Xiubo Geng, Qingwei Lin, Shifeng Chen, and Dongmei Zhang. 2023. Wizardmath: Empowering mathematical reasoning for large language models via reinforced evol-instruct. *CoRR*, abs/2308.09583.

Ziyang Luo, Can Xu, Pu Zhao, Qingfeng Sun, Xiubo Geng, Wenxiang Hu, Chongyang Tao, Jing Ma, Qingwei Lin, and Daxin Jiang. 2024. Wizardcoder: Empowering code large language models with evol-instruct. In *The Twelfth International Conference on Learning Representations, ICLR 2024, Vienna, Austria, May 7-11, 2024*. OpenReview.net.

Microsoft. 2023. Fine-tune a foundation model from the model catalog in Azure Machine Learning - Training.

Markus Nagel, Rana Ali Amjad, Mart van Baalen, Christos Louizos, and Tijmen Blankevoort. 2020. Up or down? adaptive rounding for post-training quantization. In *Proceedings of the 37th International Conference on Machine Learning, ICML 2020, 13-18 July 2020, Virtual Event*, volume 119 of *Proceedings of Machine Learning Research*, pages 7197–7206. PMLR.

OpenAI. 2023a. GPT-3.5 Turbo fine-tuning and API updates.

OpenAI. 2023b. GPT-4 technical report. *CoRR*, abs/2303.08774.

Kuntal Kumar Pal, Kazuaki Kashihara, Ujjwala Anantheswaran, Kirby C. Kuznia, Siddhesh Jagtap, and Chitta Baral. 2023. Exploring the limits of transfer learning with unified model in the cybersecurity domain. *CoRR*, abs/2302.10346.

Bowen Ping, Shuo Wang, Hanqing Wang, Xu Han, Yuzhuang Xu, Yukun Yan, Yun Chen, Baobao Chang, Zhiyuan Liu, and Maosong Sun. 2024. Delta-come: Training-free delta-compression with mixed-precision for large language models. *CoRR*, abs/2406.08903.

Mohammad Rastegari, Vicente Ordonez, Joseph Redmon, and Ali Farhadi. 2016. Xnor-net: Imagenet classification using binary convolutional neural networks. In *Computer Vision - ECCV 2016 - 14th European Conference, Amsterdam, The Netherlands, October 11-14, 2016, Proceedings, Part IV*, volume 9908 of *Lecture Notes in Computer Science*, pages 525–542. Springer.

Houxing Ren, Mingjie Zhan, Zhongyuan Wu, Aojun Zhou, Junting Pan, and Hongsheng Li. 2024. Reflectioncoder: Learning from reflection sequence for enhanced one-off code generation. *Preprint*, arXiv:2405.17057.

Baptiste Rozière, Jonas Gehring, Fabian Gloeckle, Sten Sootla, Itai Gat, Xiaoqing Ellen Tan, Yossi Adi, Jingyu Liu, Tal Remez, Jérémy Rapin, Artyom Kozhevnikov, Ivan Evtimov, Joanna Bitton, Manish Bhatt, Cristian Canton-Ferrer, Aaron Grattafiori, Wenhan Xiong, Alexandre Défossez, Jade Copet, Faisal Azhar, Hugo Touvron, Louis Martin, Nicolas Usunier, Thomas Scialom, and Gabriel Synnaeve. 2023. Code llama: Open foundation models for code. *CoRR*, abs/2308.12950.

Simo Ryu, Seunghyun Seo, and Jaejun Yoo. 2023. Efficient storage of fine-tuned models via low-rank approximation of weight residuals. *CoRR*, abs/2305.18425.

Wenqi Shao, Mengzhao Chen, Zhaoyang Zhang, Peng Xu, Lirui Zhao, Zhiqian Li, Kaipeng Zhang, Peng Gao, Yu Qiao, and Ping Luo. 2023. Omniquant: Omnidirectionally calibrated quantization for large language models. *CoRR*, abs/2308.13137.

Ying Sheng, Shiyi Cao, Dacheng Li, Coleman Hooper, Nicholas Lee, Shuo Yang, Christopher Chou, Banghua Zhu, Lianmin Zheng, Kurt Keutzer, Joseph E. Gonzalez, and Ion Stoica. 2023. S-lora: Serving thousands of concurrent lora adapters. *CoRR*, abs/2311.03285.

Amanpreet Singh, Vivek Natarjan, Meet Shah, Yu Jiang, Xinlei Chen, Devi Parikh, and Marcus Rohrbach. 2019. Towards vqa models that can read. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pages 8317–8326.

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

Xiaoguang Tu, Zhi He, Yi Huang, Zhi-Hao Zhang, Ming Yang, and Jian Zhao. 2024. An overview of large ai models and their applications. *Visual Intelligence*, 2.

Xiuying Wei, Yunchen Zhang, Xiangguo Zhang, Ruihao Gong, Shanghang Zhang, Qi Zhang, Fengwei Yu, and Xianglong Liu. 2022. Outlier suppression: Pushing the limit of low-bit transformer language models. In *NeurIPS*.

Yuxiang Wei, Zhe Wang, Jiawei Liu, Yifeng Ding, and Lingming Zhang. 2024. Magicoder: Empowering code generation with oss-instruct. In *Forty-first International Conference on Machine Learning, ICML 2024, Vienna, Austria, July 21-27, 2024*. OpenReview.net.

Guangxuan Xiao, Ji Lin, Mickaël Seznec, Hao Wu, Julien Demouth, and Song Han. 2023a. Smoothquant: Accurate and efficient post-training quantization for large language models. In *International Conference on Machine Learning, ICML 2023, 23-29 July 2023, Honolulu, Hawaii, USA*, volume 202 of *Proceedings of Machine Learning Research*, pages 38087–38099. PMLR.

Yisong Xiao, Aishan Liu, Tianyuan Zhang, Haotong Qin, Jinyang Guo, and Xianglong Liu. 2023b. Robustmq: benchmarking robustness of quantized models. *Vis. Intell.*, 1(1).

Xuyuan Xiong, Simeng Han, Ziyue Zhou, and Arman Cohan. 2024. Metamath: Integrating natural language and code for enhanced mathematical reasoning in large language models. *CoRR*, abs/2409.19381.

Can Xu, Qingfeng Sun, Kai Zheng, Xiubo Geng, Pu Zhao, Jiazhan Feng, Chongyang Tao, Qingwei Lin, and Daxin Jiang. 2024. Wizardlm: Empowering large pre-trained language models to follow complex instructions. In *The Twelfth International Conference on Learning Representations, ICLR 2024, Vienna, Austria, May 7-11, 2024*. OpenReview.net.

Xiaozhe Yao and Ana Klimovic. 2023. Deltazip: Multi-tenant language model serving via delta compression. *CoRR*, abs/2312.05215.

Le Yu, Bowen Yu, Haiyang Yu, Fei Huang, and Yongbin Li. 2024. Language models are super mario: Absorbing abilities from homologous models as a free lunch. In *Forty-first International Conference on Machine Learning, ICML 2024, Vienna, Austria, July 21-27, 2024*. OpenReview.net.

Xiang Yue, Xingwei Qu, Ge Zhang, Yao Fu, Wenhao Huang, Huan Sun, Yu Su, and Wenhu Chen. 2024. Mammoth: Building math generalist models through hybrid instruction tuning. In *The Twelfth International Conference on Learning Representations, ICLR 2024, Vienna, Austria, May 7-11, 2024*. OpenReview.net.

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 Conference of the Association for Computational Linguistics, ACL 2019, Florence, Italy, July 28- August 2, 2019, Volume 1: Long Papers*, pages 4791–4800. Association for Computational Linguistics.

Lianmin Zheng, Liangsheng Yin, Zhiqiang Xie, Chuyue Sun, Jeff Huang, Cody Hao Yu, Shiyi Cao, Christos Kozyrakis, Ion Stoica, Joseph E. Gonzalez, Clark W. Barrett, and Ying Sheng. 2024. Sglang: Efficient execution of structured language model programs. In *Advances in Neural Information Processing Systems 38: Annual Conference on Neural Information Processing Systems 2024, NeurIPS 2024, Vancouver, BC, Canada, December 10 - 15, 2024*.

A Appendix

A.1 Detailed Experiment Results

In Section 4, due to page limitations, we present only the aggregated results from two evaluation datasets. The detailed results can be found in Table 5 to Table 8.

A.2 Visualization of the RQT

Figure 6 shows a visualization of the RQT applied to the weights of the down projection layer in the first decoder layer. We observe that (a) similar models are positioned closer together in the RQT and (b) the reconstruction errors for each model decrease as the residual quantization performed from top to down. These observations validate the effectiveness of our algorithm.

A.3 More details about ablation studies

The algorithm of top-down single-step method Although we propose a two-step greedy approach for constructing an RQT, we also experimented with a single-step method in a top-down manner, as shown in Table 2. Specifically, at each node, we first optimize the quantization parameters using Equation (7) and Equation (6), and then determine how to distribute the residual weights among the child nodes by Equation (9)

$$\begin{aligned} \min_{\{x_i\}_{1\le i\le N}} \quad & \sum_{1\le i<j\le N} I(x_i = x_j)\cdot d_{ij}, \\ \text{subject to:} \quad & \sum_{1\le i\le N} x_i = \frac{N}{2}, \quad x_i \in \{0,1\}. \end{aligned} \tag{9}$$

Here, x_i is a binary variable indicating whether node i belongs to the first set. The indicator function $I(x_i = x_j)$ equals 1 when $x_i = x_j$ (W_i and W_j belong to the same set), and 0 otherwise. Additionally, d_{ij} represents the distance between the weight matrices W_i and W_j, and it is chosen as the L_2 distance:

$$d_{ij} = \|W_i - W_j\|_2. \tag{10}$$

The complete pseudocode is formally described in Algorithm 3.

The algorithm of model addition. In section 4.4, we discuss how to incorporate new models into an existing RQT without rebuilding the entire tree and relax the constraint that N is a power of 2. Specifically, we first search for the best position to add the new leaf and then perform residual quantization from the root to the leaf, only computing optimal scales for the new models without changing the shared low-bit values. The complete pseudocode is formally described in Algorithm 4.

A.4 More Ablation Studies

Root bit-width configuration. We conduct ablation studies on different bit-widths of the root node, as shown in Table 9. The results demonstrate that 4-bit quantization is sufficient to maintain performance.

A.5 Kernel Latency

Regarding the inference stage, the tree structure can be parallelized across nodes during inference, similar to the fused MoE kernel in VLLM (Kwon et al., 2023) and Sglang (Zheng et al., 2024). We modified the kernel by packing binary weights into `int32` and tested it on an NVIDIA 4090. In our experimental setup, we set the token batch size to 1 for each model, with both the input and output channels set to 4096. As shown in Table 10, our method achieves a 4× speedup compared to the naive sequential `torch.matmul` for each model.

Table 5: The detailed results of experiments on math LLMs.

MODEL	WizardMath-7B			MetaMath-7B			Xwin-Math-7B-V1.1			MuggleMath-7B		
DATASET	GSM8K	MATH	Avg.	GSM8K	MATH	Avg.	GSM8K	MATH	Avg.	GSM8K	MATH	Avg.
FP	54.20	11.30	32.75	66.71	20.80	43.76	84.53	44.42	64.48	67.10	24.90	46.00
GPTQ-w4-g128	51.30	10.00	30.65	66.60	20.10	43.35	82.90	42.18	62.54	63.80	24.00	43.90
GPTQ-W3-g128	35.90	6.10	21.00	61.00	16.70	38.85	75.57	36.46	56.02	53.40	15.30	34.35
GPTQ-W2-g128	3.10	2.70	2.90	1.10	1.63	1.37	0.37	0.42	0.40	2.00	2.30	2.15
QuaRot-w2g128	1.30	0.50	0.90	2.70	1.00	1.85	0.90	1.60	1.25	1.50	1,1	1.50
QuaRot-w3g128	40.80	6.90	23.85	61.50	17.00	39.25	76.95	36.04	56.50	58.50	17.30	37.90
Omniquant-w3g128	38.50	6.80	22.65	63.50	17.90	40.70	79.30	36.38	57.84	55.00	18.70	36.85
Omniquant-w2g128	2.30	1.60	1.95	14.30	3.20	8.75	4.01	1.70	2.86	6.70	3.20	4.95
BitDelta (1-bit Delta)	51.60	10.30	30.95	62.90	19.20	41.05	78.24	36.90	57.57	62.20	23.10	42.65
GPT-Zip (2-bit Delta)	52.80	11.40	32.10	65.30	21.50	43.40	83.62	42.80	63.21	66.00	25.40	45.70
Delt-CoME	52.2	10.7	31.45	64.7	20.3	42.5	79.83	39.56	59.695	63.3	24.4	43.85
RQT	58.20	12.70	35.45	66.00	20.70	43.35	83.01	40.92	61.97	70.80	26.10	48.45
RQT (w/o scales distillation)	57.60	13.10	35.35	65.40	20.50	42.95	82.94	40.90	61.92	69.80	25.00	47.40
RQT (w/o model-specific scales)	59.10	12.80	35.95	66.20	20.40	28.30	81.12	40.16	60.64	70.80	26.00	48.40
Top-down Single-step method	49.70	10.10	29.90	65.40	20.20	29.30	78.92	36.06	57.49	65.70	23.20	44.45
Cosine Similarity	57.50	13.00	35.25	65.00	21.30	30.30	83.01	41.88	62.45	68.70	23.80	46.25
Sign difference	47.90	10.10	29.00	66.60	20.30	31.30	83.26	41.52	62.39	70.40	24.90	47.65
2-bit Root	0	0	0	0	0	0	1.21	0.7	0	0	0	0
3-bit Root	55.2	11.7	11.7	65.2	19.9	42.55	81.42	39.66	60.54	69.1	24.9	47
8-bit Root	59.4	13.3	36.35	65.8	21.2	43.5	83.69	42.24	62.97	68.6	25.7	47.15

Table 6: The detailed experimental results on code LLMs are provided. HE refers to the HumanEval (Chen et al., 2021) dataset.

MODEL	CodeLLaMA-Python-7B			WizardCoder-7B			Magicoder-S-CL-7B			ReflectionCoder-7B		
DATASET	HE	MBPP	Avg.	HE	MBPP	Avg.	HE	MBPP	Avg.	HE	MBPP	Avg.
FP	40.84	43.80	42.32	48.78	64.55	56.67	69.50	69.60	69.55	74.40	71.69	73.05
GPTQ-w4-g128	34.75	42.60	38.68	42.07	61.11	51.59	72.00	68.80	70.40	73.17	67.46	70.32
GPTQ-W3-g128	22.56	24.40	23.48	29.87	47.35	38.61	51.80	59.30	55.55	56.09	58.99	57.54
GPTQ-W2-g128	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00
QuaRot-w2g128	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00
QuaRot-w3g128	29.87	34.40	32.14	33.53	52.38	42.96	60.40	61.60	61.00	61.58	65.87	63.73
Omniquant-w3g128	31.20	30.48	30.84	37.80	54.76	46.28	64.00	61.40	62.70	60.36	65.60	62.98
Omniquant-w2g128	6.09	2.20	4.15	2.43	0.79	1.61	7.90	27.50	17.70	15.24	22.22	18.73
BitDelta (1-bit Delta)	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00
GPT-Zip (2-bit Delta)	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00
Delt-CoME	50	0	0	0	0	0	0	0	0	0	0	0
RQT	33.53	29.60	31.57	42.07	60.31	51.19	62.80	65.30	64.05	64,63	62.43	62.43
RQT (w/o scales distillation)	35.36	33.60	34.48	38.41	57.14	47.78	54.30	64.00	59.15	60.97	62.16	61.57
RQT (w/o model-specific scales)	43.32	15.85	20.80	18.33	7.31	34.65	20.98	27.40	46.80	37.10	37.80	52.64
Top-down Single-step method	40.29	30.60	34.14	32.37	34.14	56.08	45.11	54.90	60.80	57.85	60.97	61.90
Cosine Similarity	43.56	32.31	32.40	32.36	38.41	55.82	47.12	58.50	63.80	61.15	61.58	64.02
Sign difference	42.59	32.31	32.60	32.46	38.41	56.08	47.25	58.50	64.00	61.25	62.19	64.28
2-bit	27.43	30.2	28.815	25	49.26	37.13	48.8	60.6	54.7	50.6	62.9	63.24
3-bit	33.53	38	35.77	29.87	55.82	42.85	57.3	63.2	60.25	54.87	62.69	58.78
8-bit	31.09	39.2	35.15	28.65	57.4	43.03	59.8	63.8	61.8	62.19	62.43	62.31

Table 7: The detailed results of experiments on chat LLMs are presented. ARC-E refers to the ARC-Easy (Clark et al., 2018) dataset, and HS refers to the HellaSwag (Zellers et al., 2019) dataset.

MODEL	LLaMA-2-chat-7B			Vicuna-7B v1.5			Vicuna-7B-v1.5-16k			LLaMA-2-7B		
DATASET	ARC-E	HS	Avg.	ARC-E	HS	Avg.	ARC-E	HS	Avg.	ARC-E	HS	Avg.
FP	73.91	75.47	74.69	75.63	73.81	74.72	76.18	74.31	75.25	76.35	76.00	76.18
GPTQ-w4-g128	72.77	74.69	73.73	75.34	73.20	74.27	76.26	73.07	74.67	75.84	75.62	75.73
GPTQ-W3-g128	67.30	71.44	69.37	72.22	69.68	70.95	69.99	69.20	69.60	72.31	72.06	72.19
GPTQ-W2-g128	25.29	27.23	26.26	32.66	32.14	32.40	29.59	27.65	28.62	31.52	31.22	31.37
QuaRot-w2g128	29.04	29.08	29.06	35.73	32.24	33.99	29.67	27.96	28.82	31.48	30.55	31.02
QuaRot-w3g128	68.81	71.17	69.99	73.02	69.98	71.50	69.87	63.74	66.81	69.44	69.85	69.65
Omniquant-w3g128	69.57	71.43	70.50	72.64	70.30	71.47	72.58	69.25	70.92	74.07	72.54	73.31
Omniquant-w2g128	43.48	42.73	43.11	53.75	47.14	50.45	37.37	37.92	37.65	52.95	51.40	52.18
QuIP#	69.50	70.79	70.15	-	-	-	-	-	-	69.60	71.84	70.72
AQLM	-	-	-	-	-	-	-	-	-	72.25	72.85	72.68
BitDelta (1-bit Delta)	73.95	75.44	74.70	75.80	74.40	75.10	68.48	57.92	63.20	76.35	76.00	76.18
GPT-Zip (2-bit Delta)	74.45	75.66	75.06	76.18	74.11	75.15	76.68	74.56	75.62	76.35	75.98	76.17
Delta-CoME	74.24	75.78	75.01	76.26	74.3	75.28	76.01	74.67	75.34	76.35	76	76.175
RQT	74.03	75.80	74.92	76.47	74.32	75.40	75.42	74.15	74.79	74.62	75.93	75.28
RQT (w/o scales distillation)	74.33	74.98	74.66	76.22	73.70	74.96	75.04	73.57	74.31	75.04	76.08	75.56
RQT (w/o model-specific scales)	45.22	30.41	73.86	74.20	74.03	75.88	73.04	74.46	74.96	72.60	73.78	75.93
Top-down Single-step method	61.44	49.19	74.83	74.78	74.81	75.38	73.17	74.28	76.09	72.89	74.49	75.55
Cosine Similarity	62.80	50.86	75.00	74.92	74.96	75.97	73.47	74.72	75.08	73.61	74.35	75.76
Sign difference	63.24	51.05	74.33	74.16	74.25	75.88	73.56	74.72	75.13	73.63	74.38	75.63
2-bit	25.88	25.81	25.845	26.14	25.59	25.865	26.26	25.96	26.11	26.18	25.88	26.03
3-bit	74.62	73.14	73.88	74.45	71.24	72.85	74.54	69.63	72.09	76.09	73.99	75.04
8-bit	75	75.78	75.39	76.18	74.36	75.27	76.18	74.04	75.11	76.05	76.09	76.07

Table 8: The detailed results of experiments on Chinese LLMs.

MODEL	Chinese-Alpaca-2			Chinese-LLAMA-2			Chinese-Alpaca-2-16k			Chinese-LLAMA-2-16k		
DATASET	CEval	CMMLU	Avg.	CEval	CMMLU	Avg.	CEval	CMMLU	Avg.	CEval	CMMLU	Avg.
FP	42.05	39.88	40.97	28.45	27.66	28.06	37.89	37.18	37.54	28.45	26.52	27.49
GPTQ-w4-g128	41.08	39.02	40.05	26.52	27.29	26.91	37.41	35.71	36.56	25.92	25.75	25.84
GPTQ-W3-g128	34.99	33.84	34.42	26.52	26.04	26.28	35.36	33.69	34.53	23.32	25.28	24.30
GPTQ-W2-g128	24.14	24.71	24.43	23.84	24.57	24.21	26.37	25.09	25.73	28.15	25.52	26.84
QuaRot-w2g128	25.70	24.80	25.25	25,85	24.90	24.90	25.18	25.03	25.11	22.88	25.19	24.04
QuaRot-w3g128	32.76	33.05	32.91	26.59	25.84	26.22	36.55	32.49	34.52	26.96	24.98	25.97
Omniquant-w3g128	35.14	32.43	33.79	26.82	25.83	26.33	35.21	32.24	33.73	26.74	25.87	26.31
Omniquant-w2g128	23.25	25.41	24.33	23,03	25.27	25.27	23,47	24.72	24.72	22.95	25.40	24.18
BitDelta (1-bit Delta)	38.11	35.70	36.91	25.92	27.01	26.47	30.98	27.28	29.13	24.96	25.66	25.31
GPT-Zip (2-bit Delta)	41.60	38.05	39.83	26.98	26.86	26.92	35.51	35.60	35.56	25.85	26.40	26.13
Delta-CoME	39.99	38.39	39.19	27.93	26.67	27.3	38.7	35.22	36.96	26.22	26.29	26.255
RQT	41.30	39.46	40.38	27.86	27.39	27.63	36.62	35.52	36.07	27.34	27.05	27.20
RQT (w/o scales distillation)	40.71	37.94	39.33	28.52	28.58	28.55	37.29	35.10	36.20	27.56	27.54	27.55
RQT (w/o model-specific scales)	75.17	75.55	74.46	40.56	37.30	38.93	27.34	27.14	27.24	36.47	34.53	35.50
Top-down Single-step method	75.50	75.53	74.77	35.58	35.35	35.47	26.96	26.54	26.75	34.99	32.99	33.99
Cosine Similarity	75.80	75.78	74.95	38.03	36.04	37.04	28.08	28.03	28.06	36.47	34.38	35.43
Sign difference	76.09	75.86	74.80	38.03	36.04	37.04	28.08	28.03	28.06	36.47	34.38	35.43
2-bit	23.03	25.26	24.145	23.03	25.26	24.145	23.03	25.26	24.145	23.03	25.26	24.145
3-bit	37.07	36.68	36.88	27.11	27.05	27.08	35.21	35.27	35.24	26	26.27	26.135
8-bit	41.45	38.86	40.16	28	28.25	28.13	36.99	35.65	36.32	29.49	27.55	28.52

Bit	WizardMath	MetaMath	Xwin-Math	MuggleMath	CODELLAMA-PY	wizardcoder-python	Magicoder-S-CL	ReflectionCoder-7B
FP	32.75	43.755	64.475	46.00	42.32	56.665	69.55	73.045
2-bit	0.00	0.00	0.00	0.00	28.815	37.130	54.70	63.240
3-bit	11.70	42.55	60.54	47.00	35.770	42.850	60.25	58.780
4-bit	35.35	42.95	61.92	47.40	34.480	47.775	59.15	61.565
8-bit	36.35	43.50	62.97	47.15	35.150	43.030	61.80	62.310

Bit	LLAMA-2-CHAT	Vicuna-7B v1.5	vicuna-7b-v1.5-16k	Llama-2-7b-hf	Chinese-Alpaca-2	Chinese-LLAMA-2	Chinese-Alpaca-2-16k	Chinese-LLAMA-2-16k
FP	74.69	74.72	75.245	76.175	40.965	28.055	37.535	27.485
2-bit	25.845	25.865	26.110	26.030	24.145	24.145	24.145	24.145
3-bit	73.880	72.850	72.090	75.040	36.880	27.080	35.240	26.135
4-bit	74.655	74.960	74.305	75.560	39.325	28.550	36.195	27.550
8-bit	75.390	75.270	75.110	76.070	40.160	28.130	36.320	28.520

Table 9: Ablation studies on bit-width of the root node.

Algorithm 3 Single_Step_Method

Input: A set of weights $\Theta = \{W_i \mid i = 1, \ldots, N\}$
Output: A complete residual quantization tree T with defined Q and Σ
while not converged **do**
    Compute α_i using Equation 6
    Compute Q using Equation 7
end while
Update $W_i \leftarrow W_i - Q \cdot \alpha_i$ for all i
Create child nodes $T_{\text{left}}, T_{\text{right}}$
Set $C[T] \leftarrow \{T_{\text{left}}, T_{\text{right}}\}$
Compute $S[T_{\text{right}}]$ and $S[T_{\text{left}}]$ using Equation 9
for each $T_j \in C[T]$ **do**
    Single_Step_Method($\{W_i \mid i \in S[T_j]\}, T_j$)
end for
return T

Algorithm 4 Model_Addition

Input: A set of weights $\Theta = \{W_i \mid i = 1, \ldots, N\}$
Input: Newly added weight W_{N+1}
Input: A residual quantization tree T built on Θ
$k = \arg\min ||W_{N+1} - W_j||$ for $j \in \{1, \ldots, N\}$
Let T_k denote the leaf node where $S[T_k] = \{k\}$
while true **do**
    Compute $\alpha_{\text{N+1}}$ by Equation (6)
    Set $\Sigma[T] \leftarrow \Sigma[T] \cup \{\alpha_{\text{N+1}}\}$
    Set $S[T] \leftarrow S[T] \cup \{N+1\}$
    Set $W_{\text{N+1}} \leftarrow W_{\text{N+1}} - \alpha_{\text{N+1}} \cdot Q[T]$
    if T = parent(T_k) **then**
        break
    else
        $T = T_j$ where $T_j \in C[T]$ and $k \in T_j$
    end if
end while
Create a new leaf node T_{new}
$S[T_{\text{new}}] = \{N+1\}$
$C[\text{parent}(T_k)] = C[\text{parent}(T_k)] \cup \{T_{\text{new}}\}$
Compute $Q[T_{\text{new}}]$ and $\alpha_{\text{N+1}}[T_{\text{new}}]$
return T

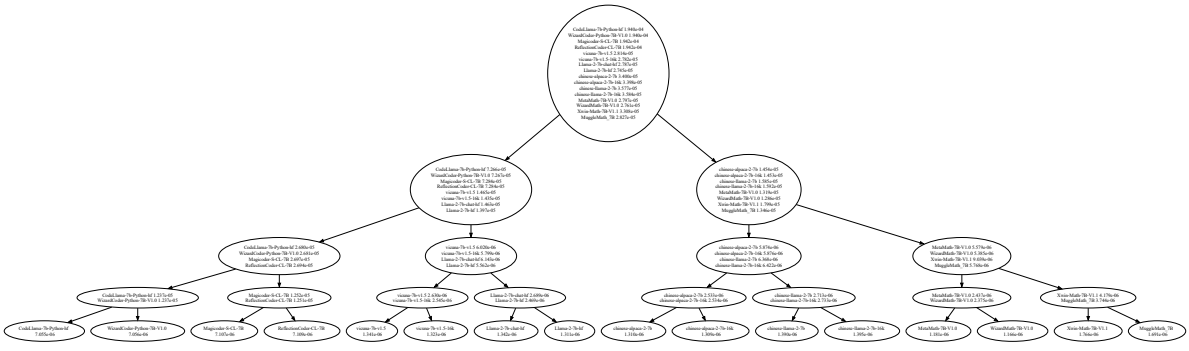


Figure 6: Visualization of the Residual Quantization Tree (RQT) applied to the down projector of the first decoder layer, with the MSE of each model annotated.

Table 10: Latency of computation kernel of RQT

Model Number	**Torch (ms)**	**RQT (ms)**	**Speedup**
4	0.2795	0.0675	4.14
8	0.5386	0.1443	3.73
16	1.0639	0.2672	3.98
32	2.0987	0.5611	3.74