ApiQ: Finetuning of 2-Bit Quantized Large Language Model

Baohao Liao[1,2*] **Christian Herold**[2] **Shahram Khadivi**[2] **Christof Monz**[1]
[1]Language Technology Lab, University of Amsterdam
[2]eBay Inc., Aachen, Germany
Code: https://github.com/BaohaoLiao/ApiQ

Abstract

Memory-efficient finetuning of large language models (LLMs) has recently attracted huge attention with the increasing size of LLMs, primarily due to the constraints posed by GPU memory limitations and the effectiveness of these methods compared to full finetuning. Despite the advancements, current strategies for memory-efficient finetuning, such as QLoRA, exhibit inconsistent performance across diverse bit-width quantizations and multifaceted tasks. This inconsistency largely stems from the detrimental impact of the quantization process on preserved knowledge, leading to catastrophic forgetting and undermining the utilization of pretrained models for finetuning purposes. In this work, we introduce a novel quantization framework named *ApiQ*, designed to restore the lost information from quantization by concurrently initializing the LoRA components and quantizing the weights of LLMs. This approach ensures the maintenance of the original LLM's activation precision while mitigating the error propagation from shallower into deeper layers. Through comprehensive evaluations conducted on a spectrum of language tasks with various LLMs, ApiQ demonstrably minimizes activation error during quantization. Consequently, it consistently achieves superior finetuning results across various bit-widths. Notably, one can even finetune a 2-bit Llama-2-70b with ApiQ on a single NVIDIA A100-80GB GPU without any memory-saving techniques, and achieve promising results.[1]

1 Introduction

Large language models (LLMs) have garnered significant acclaim and success across a wide range of domains and applications (Touvron et al., 2023b; Jiang et al., 2023; OpenAI, 2023). With ongoing advancements, the scope and complexity of released LLMs have witnessed exponential growth, with some LLMs encompassing more than 50B parameters (Touvron et al., 2023b,a; Zhang et al., 2022; Scao et al., 2022). This remarkable upscaling introduces considerable challenges, particularly when effectively adapting these models for downstream tasks. Historically, a prevalent method for adapting pretrained models to specific downstream tasks is full finetuning, a process that updates all pretrained parameters. Although this approach has led to many state-of-the-art achievements, its practicality is somewhat hindered in scenarios where GPU memory is limited. This limitation stems from the necessity to store the model's weights and optimizer's states in the GPU's memory and intensifies with the escalating sizes of LLMs.

To mitigate the extensive memory requirement for finetuning LLMs, an alternative method is parameter-efficient finetuning (PEFT) (Liao et al., 2023a; Hu et al., 2022; Houlsby et al., 2019). This technique involves selectively introducing and updating a limited set of parameters while leaving the majority of the LLM's parameters unchanged. A key advantage of this approach is the substantial reduction in GPU memory required for the optimizer's states since the size of the optimizer states is proportional to the amount of trainable parameters. To further reduce the GPU memory required by loading the LLM's weights, multiple model compression methods have been proposed (Shao et al., 2023a; Xiao et al., 2023; Dettmers et al., 2023b; Lin et al., 2023; Frantar et al., 2022), converting high-precision weight values into a discrete set of values. Initially, quantization techniques were primarily developed for deploying LLMs in memory-limited environments for inference purposes. QLoRA (Dettmers et al., 2023a) innovatively combines PEFT, specifically LoRA (Hu et al., 2022), with quantization methods to remarkably reduce the GPU memory requirement for finetuning.

However, QLoRA (Dettmers et al., 2023a) in-

*Correspondence to: b.liao@uva.nl
[1]Please read the newest version at https://arxiv.org/abs/2402.05147.

Proceedings of the 2024 Conference on Empirical Methods in Natural Language Processing, pages 20996–21020
November 12-16, 2024

herits the same challenge as LLM's quantization, namely the distortion of the learned knowledge from the full-precision LLM due to the quantization error, which exacerbates at lower-bit quantizations, leading to catastrophic forgetting. Recently, Li et al. (2023) and Guo et al. (2023) proposed a new method to reduce the quantization error through a strategic initialization of the LoRA components to maintain the original weight states. This technique has demonstrated considerable success in the finetuning of lower-bit quantized LLMs (QLLMs). Nonetheless, they focus on preserving the weight states on a per-linear-layer basis, resulting in accumulative error propagation through layers.

In this paper, we introduce a novel and efficient quantization framework, termed *ApiQ*, which consists of two steps to adapt an LLM, similar to QLoRA. During the quantization step, ApiQ preserves the activation instead of the weight of full-precision LLM by jointly optimizing the LoRA's components and quantizing the LLM's weights. This approach ensures that the output of the QLLM remains consistent with that of the full-precision LLM, effectively mitigating quantization error by aligning the activations across corresponding layers. As a result, the quantization errors introduced in earlier layers are ameliorated. Subsequently, we finetune the LoRA modules with the fixed QLLM on downstream tasks, thereby significantly reducing the demands on GPU memory.

Our primary contributions are as follows:

- We conduct an in-depth analysis of the challenges associated with finetuning QLLM (§3).
- We propose ApiQ to initialize the PEFT parameters in conjunction with the quantization of an LLM, aiming to retain the activation of the full-precision LLM. ApiQ demonstrates superior performance post-quantization, even surpassing the latest post-training quantization (PTQ) techniques (§4).
- We carry out extensive finetuning experiments on 5 LLMs across 5 different tasks to evaluate the effectiveness of ApiQ. ApiQ consistently outperforms all baselines at various bit levels (§5 and Figure 4).

2 Preliminaries

GPU memory allocation and utilization are typified by three principal mechanisms for training a model, as exemplified in Figure 1 during the training of Llama-2-7b (Touvron et al., 2023b). Initially, a substantial portion of GPU memory is allocated to store the model's weights. For instance, approximately 12.6GB is required for a model comprising roughly 7B parameters in BFloat16 format. Secondly, the optimizer states associated with trainable parameters occupy a considerable amount of GPU memory. Employing Adam (Kingma and Ba, 2015) as the optimizer necessitates storing the first and second moments in the GPU memory, effectively doubling the memory requirement compared to that needed for the trainable parameters alone. Notably, the memory allocations for the model's weights and optimizer states are static, remaining constant throughout the training process. The third aspect involves the temporary storage of *activations* — the outputs produced by each layer as data traverses through the model. These activations are crucial for gradient computation during the backward pass and are retained in memory for this purpose. After the gradient computation, these activations are discarded. Modern training frameworks, like PyTorch (Paszke et al., 2019), employ a sequential process for gradient computation and activation deletion, enhancing memory efficiency. Subsequently, gradients are utilized to update the model's weights and optimizer states, and then they too are eliminated. Peak memory usage typically occurs at the onset of gradient computation or during the update of optimizer states.

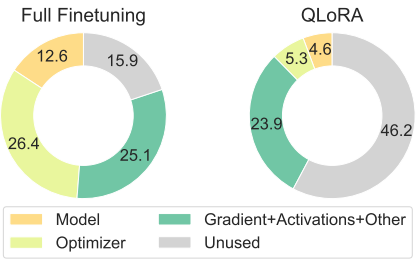

Figure 1: Memory allocation (GB) of a A100-80GB GPU for finetuning Llama-2-7b. The optimizer is Adam. The batch size is 1. The sequence length is 2048. For QLoRA, the bit-width is 4 and the LoRA rank is 64.

Memory-efficient finetuning. In response to the GPU memory constraints and the increasing size of LLMs, various strategies have been developed to optimize memory efficiency during finetuning. To mitigate activation memory demands, techniques such as selective activation storage and on-demand recomputation are employed (Liao et al., 2023b; Gomez et al., 2017; Chen et al., 2016). Ad-

ditionally, to curtail the memory required for optimizer state storage, the pretrained LLM is kept fixed while a limited amount of trainable parameters are introduced (Hu et al., 2022; Houlsby et al., 2019). A prime example is LoRA (Hu et al., 2022), which adapts the pretrained weight, $\boldsymbol{W} \in \mathbb{R}^{d_1 \times d_2}$, of a linear layer as $\boldsymbol{W}' = \boldsymbol{W} + \frac{\alpha}{r}\boldsymbol{A}\boldsymbol{B}^\top$, where $\boldsymbol{A} \in \mathbb{R}^{d_1 \times r}$, $\boldsymbol{B} \in \mathbb{R}^{d_2 \times r}$, $r \ll d_1$, $r \ll d_2$ and α is a scalar. Introducing a smaller bottleneck dimension r substantially reduces the memory demand for the optimizer, illustrated by a reduction from 26.4GB to 5.3GB as shown in Figure 1. To further diminish LoRA's memory usage, Dettmers et al. (2023a) introduced a quantized version of $\boldsymbol{W}$, such as a 4-bit representation in contrast to 16 bits. This technique significantly decreases the memory requirement for storing the model's weights, from 12.6GB to 4.6GB.

Quantization involves converting high-precision values into discrete levels. In this research, we focus on uniform affine quantization (Jacob et al., 2018), known for its enhanced hardware support and efficiency. This process quantizes the pretrained weight as follows:

$$\boldsymbol{W}_q = \text{clamp}(\lfloor \frac{\boldsymbol{W}}{s} \rceil + z, 0, 2^b - 1) \qquad (1)$$

where the scale factor $s = \frac{\max(\boldsymbol{W}) - \min(\boldsymbol{W})}{2^b - 1}$, the zero-point $z = -\lfloor \frac{\min(\boldsymbol{W})}{s} \rceil$, b is the bit-width, and $\lfloor \rceil$ is the round-to-nearest operation. One only needs to load $\boldsymbol{W}_q$ and z in a reduced bit format, and s in Float16 to GPU. During the forward pass, they are de-quantized for activation computation as $\boldsymbol{Q} = s(\boldsymbol{W}_q - z)$.

3 Challenges of Finetuning QLLM

QLoRA (Dettmers et al., 2023a) employs a strategy wherein the fixed pretrained weights are loaded onto the GPU in a lower-bit format, while finetuning is confined to a minimal number of parameters from the adapters. This approach significantly reduces the memory allocation required from both the model's weights and optimizer states, decreasing it from 39GB to 10GB, as depicted in Figure 1. This reduction in memory demand facilitates the finetuning of LLMs on more modest computational resources. Nevertheless, this method introduces certain challenges associated with quantization.

3.1 QLLM breaks down the starting point

LLMs are recognized for their ability to learn broadly applicable and distributed representations that effectively support the downstream learning of compressed task-specific representations (Aghajanyan et al., 2021), i.e. offering a robust starting point for the training of downstream tasks. Liao et al. (2023b) postulate that maintaining this starting point — ensuring that the difference between the modified weight $\boldsymbol{W}'$ and the original weight $\boldsymbol{W}$ is minimal (i.e., $\|\boldsymbol{W}' - \boldsymbol{W}\| \to 0$) — is crucial at the finetuning's outset to achieve performance comparable to full finetuning.

Method	LoRA position	MNLI (acc↑) 2 Bits	WikiText (ppl↓) 4 Bits	WikiText (ppl↓) 2 Bits
QLoRA	All	**79.7**	**5.24**	N.A.
	FFN	78.2	5.29	N.A.
	Attn	75.7	5.28	N.A.
LoftQ	All	**88.5**	**5.24**	**7.85**
	FFN	87.1	5.30	8.64
	Attn	87.5	5.28	8.86
ApiQ-lw	All	**88.6**	5.28	**7.46**
	FFN	88.2	5.29	7.50
	Attn	**88.6**	**5.25**	7.55

Table 1: The effect of trainable LoRA position. All linear layers are incorporated with a LoRA module initialized with different methods. Only the LoRA modules in the denoted position are finetuned. ApiQ has the smallest gap between different positions.

LoRA (Hu et al., 2022) adheres to this principle by initializing $\boldsymbol{B} = \boldsymbol{0}$, which results in $\boldsymbol{W}' = \boldsymbol{W}$ at the start of the training. QLoRA (Dettmers et al., 2023a), on the other hand, follows LoRA's default initialization for $\boldsymbol{A}$ and $\boldsymbol{B}$. Consequently, at the onset of training, $\boldsymbol{W}' = \boldsymbol{Q} + \boldsymbol{A}\boldsymbol{B}^\top = \boldsymbol{Q}$. Due to the round-to-nearest and clipping operations involved in quantization, $\boldsymbol{Q}$ differs from the original $\boldsymbol{W}$, thereby distorting the starting point. This deviation, represented by $\|\delta\boldsymbol{W}\| = \|\boldsymbol{W} - \boldsymbol{W}'\|$, is expected to increase with lower-bit quantization.

Recent developments by Li et al. (2023) and Guo et al. (2023) introduced an approach to initialize the $\boldsymbol{Q}$, $\boldsymbol{A}$ and $\boldsymbol{B}$ matrices in QLoRA by solving the following optimization problem:

$$\underset{\boldsymbol{Q},\boldsymbol{A},\boldsymbol{B}}{\text{argmin}} \|\boldsymbol{W} - (\boldsymbol{Q} + \boldsymbol{A}\boldsymbol{B}^\top)\| \qquad (2)$$

The key objective of this technique is to obtain $\boldsymbol{Q}$, $\boldsymbol{A}$, and $\boldsymbol{B}$ in such a way that the initial state of the model (the starting point) is preserved as closely as possible. As shown in Figure 2 (Left), LoftQ (Li et al., 2023) significantly reduces the weight error.

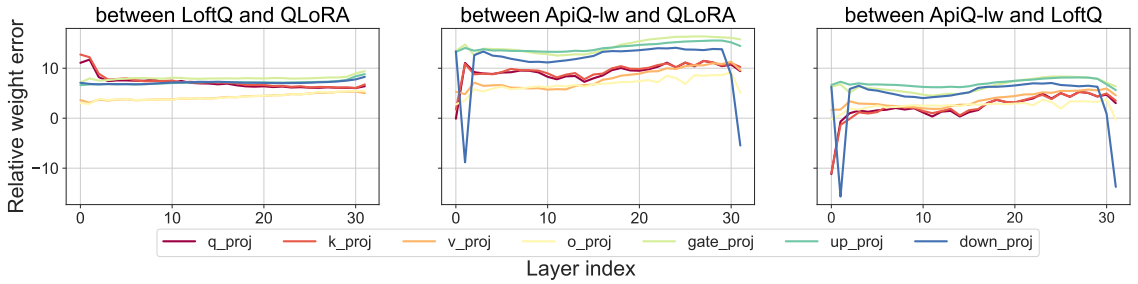

Figure 2: Relative weight quantization error of 2-bit quantized Llama-2-7b, i.e. $e = \|\delta W^{\text{baseline}}\|_F - \|\delta W^{\text{method}}\|_F$. The larger e is, the more effective the method is in reducing weight error compared to the baseline. **Left**: The method is LoftQ and the baseline is QLoRA. **Middle**: The method is ApiQ and the baseline is QLoRA. **Right**: The method is ApiQ and the baseline is LoftQ. Refer to Figure C.1 for the 2-bit and 4-bit non-relative weight error.

3.2 Accumulative quantization error

The findings of Hu et al. (2022) highlight that integrating LoRA modules solely into the query and value projection layers is adequate to match the performance of full finetuning. This stands in contrast to the stance of Dettmers et al. (2023a), who advocate for integrating the LoRA modules into all linear layers of QLLM to achieve similar results.

We extend upon the conclusion of QLoRA by conducting finetuning experiments with DeBERTa-v3-base (He et al., 2023) and Llama-2-7b (Touvron et al., 2023b) on the MNLI (Williams et al., 2018) and WikiText-2 (Merity et al., 2017) datasets, respectively. As presented in Table 1, the most effective results from QLoRA are achieved when the LoRA modules in all linear layers are trained, rather than a subset of them. This observation leads us to propose that each linear layer undergoes a loss of learned information as a consequence of quantization. To mitigate this loss and restore the original learned information, it is essential to adapt each linear layer individually with a LoRA module.

Furthermore, we notice that the quantization errors accumulate through layers. Consider two consecutive linear layers, $\boldsymbol{W}_0$ and $\boldsymbol{W}_1$, with inputs and outputs $\boldsymbol{X}_0$, $\boldsymbol{X}_1$ and $\boldsymbol{X}_2$, respectively. Under QLoRA's quantization, the activation error for the first layer is $\|\boldsymbol{X}_1 - \boldsymbol{X}_1^q\| = \|\boldsymbol{X}_0\boldsymbol{W}_0 - \boldsymbol{X}_0\boldsymbol{Q}_0\| = \|\boldsymbol{X}_0\boldsymbol{W}_0 - \boldsymbol{X}_0(\boldsymbol{W}_0 - \delta\boldsymbol{W}_0)\| = \|\boldsymbol{X}_0\delta\boldsymbol{W}_0\|$, where $\delta\boldsymbol{W}_0$ is the quantization error. For the second layer, the error is $\|\boldsymbol{X}_2 - \boldsymbol{X}_2^q\| = \|\boldsymbol{X}_0\boldsymbol{W}_0\delta\boldsymbol{W}_1 + \boldsymbol{X}_0\delta\boldsymbol{W}_0\boldsymbol{W}_1 - \boldsymbol{X}_0\delta\boldsymbol{W}_0\delta\boldsymbol{W}_1\|$. This phenomenon indicates that the quantization errors from shallower layers are propagated to deeper layers, with potentially greater impact in deeper LLMs. This effect underscores another justification for the implementation of LoRA modules in every quantized linear layer to timely counteract the errors.

Despite the advances made by LoftQ (Li et al., 2023) in reducing the quantization error $\delta\boldsymbol{W} = \boldsymbol{W} - (\boldsymbol{Q} + \boldsymbol{A}\boldsymbol{B}^\top)$, the issue of error propagation persists. This is evidenced in Table 1, where the performance between training all LoRA modules and only training a subset of them is still large, especially for lower-bit quantization. Such findings emphasize the importance of not only minimizing the quantization errors at their source but also managing their propagation across layers.

3.3 SVD is not a universal solution

Li et al. (2023) and Guo et al. (2023) apply an iterative algorithm to solve Equation (2) as:

$$\boldsymbol{A}^{(t)}, \boldsymbol{B}^{(t)} \leftarrow \text{SVD}(\boldsymbol{W} - \boldsymbol{Q}^{(t-1)})$$
$$\boldsymbol{Q}^{(t)} \leftarrow f(\boldsymbol{W} - \boldsymbol{A}^{(t)}\boldsymbol{B}^{(t)\top})$$

where f is a function for a sequential quantization and de-quantization as:[2]

$$\begin{aligned}\boldsymbol{Q} &= f(\boldsymbol{W}) \\ &= s \cdot (\text{clamp}(\lfloor \frac{\boldsymbol{W}}{s} \rceil + z, 0, 2^b - 1) - z) \quad (3)\end{aligned}$$

Although this algorithm is effective without the usage of calibration data, we couldn't easily apply it to other PEFT methods, even a variant of LoRA, i.e. DoRA (Liu et al., 2024). This algorithm requires a relationship of addition between the PEFT parameters and $\boldsymbol{W}$, which is not possible for some PEFT methods, like (IA)^3 (Liu et al., 2022a), Adapter (Houlsby et al., 2019), HiWi (Liao et al., 2023a) and so on.

[2]We use uniform affine quantization to represent the quantization of LoftQ and LQ-LoRA for easy understanding. LoftQ and LQ-LoRA actually apply NF-quantization as QLoRA.

Overall, to effectively finetune a QLLM, we need to preserve the starting point, mitigate the propagation of quantization error, and design a universal algorithm for various PEFT methods.

4 Method: ApiQ

In this section, we introduce a novel quantization framework, Activation-preserved initialization of QLLM termed as *ApiQ*, that addresses all above-mentioned challenges when finetuning a QLLM. The core objective of ApiQ is to maintain the integrity of the starting point, while effectively minimizing the cumulative impact of the quantization errors as they traverse through the network.

4.1 Activation-preserved initialization

The ApiQ framework introduces a distinct approach to quantization by focusing on minimizing the activation error, rather than the weight error as in previous methods (Li et al., 2023; Guo et al., 2023). The core optimization problem of ApiQ is formulated as follows:

$$\underset{\boldsymbol{Q},\boldsymbol{A},\boldsymbol{B}}{\operatorname{argmin}} ||\boldsymbol{X}\boldsymbol{W} - \boldsymbol{X}^q(\boldsymbol{Q} + \boldsymbol{A}\boldsymbol{B}^T)|| \quad (4)$$

The pretrained weight $\boldsymbol{W} \in \mathbb{R}^{d_1 \times d_2}$ remains fixed during optimization. The quantized weight $\boldsymbol{Q} \in \mathbb{R}_b^{d_1 \times d_2}$ is represented in b-bit format, while $\boldsymbol{A} \in \mathbb{R}^{d_1 \times r}$ and $\boldsymbol{B} \in \mathbb{R}^{d_2 \times r}$ are low-rank matrices that are trainable. The input to the linear layer $\boldsymbol{W}$ is denoted as $\boldsymbol{X} \in \mathbb{R}^{n \times t \times d_1}$, where n is the batch size and t is the sequence length. Consequently, $\boldsymbol{X}\boldsymbol{W}$ represents the output or activation of the linear layer. The input to the quantized linear layer with LoRA is $\boldsymbol{X}^q \in \mathbb{R}^{n \times t \times d_1}$. It's important to note that for the first linear layer of an LLM, $\boldsymbol{X}$ equals $\boldsymbol{X}^q$. For subsequent layers, $\boldsymbol{X}^q$ is the output from the nearest shallower quantized layer of $\boldsymbol{W}$.

A key difference from LoftQ (Li et al., 2023) and LQ-LoRA (Guo et al., 2023) is that ApiQ requires sequential optimization for each linear layer following the input order of different layers, as $\boldsymbol{X}_q$ is derived from the preceding layer. E.g., in each transformer block of Llama-2 (Touvron et al., 2023b), the optimization should start with the key, query, and value projection layers, followed by the output projection layer, then the gate and up projection layer, and finally the down projection layer.

ApiQ has two primary advantages. Firstly, it ensures that the output from the quantized linear layer closely aligns with the original output, thereby preserving the starting point of the model. Secondly, it potentially mitigates the quantization error from shallower layers into deeper layers. This is achieved by consistently enforcing that the output of each quantized layer closely matches the original output, thereby gradually easing the quantization error as it propagates through the network. This mechanism isn't present in LoftQ and LQ-LoRA, giving ApiQ a unique advantage in managing and reducing the quantization errors in QLLMs.

4.2 Block-wise ApiQ

We define Equation (4) as *layer-wise ApiQ* (*ApiQ-lw*), because the LLM is quantized in a layer-by-layer manner. Additionally, we can optimize the entire transformer block simultaneously as follows:

$$\underset{\boldsymbol{Q}s,\boldsymbol{A}s,\boldsymbol{B}s}{\operatorname{argmin}} ||\mathcal{F}(\boldsymbol{W}s, \boldsymbol{X}) - \mathcal{F}(\boldsymbol{Q}s, \boldsymbol{A}s, \boldsymbol{B}s, \boldsymbol{X}^q)||$$

where $\mathcal{F}$ denotes the mapping function of a transformer block, $\boldsymbol{W}$s represent all the weights of the linear layers within this block, $\boldsymbol{X}$ is the input to this block, $\boldsymbol{Q}$s are the quantized versions of $\boldsymbol{W}$s, $\boldsymbol{A}$s and $\boldsymbol{B}$s are all low-rank matrices within this block, and $\boldsymbol{X}^q$ is the input to the quantized block and the output from the preceding quantized block. Block-wise ApiQ (*ApiQ-bw*) necessitates sequential optimization but on a block-by-block basis, meaning we first optimize the first transformer block, followed by the second block, and so on.

ApiQ-bw retains the benefits of ApiQ-lw while offering two additional advantages. Firstly, ApiQ-bw is more time-efficient than ApiQ-lw because it quantizes the entire block in one step. Secondly, ApiQ-bw is compatible with a broader range of PEFT methods without necessitating adaptations for every linear layer. The matrices $\boldsymbol{A}$s and $\boldsymbol{B}$s do not have to be the low-rank matrices from LoRA; they can be trainable parameters from any PEFT method, such as DoRA, $(IA)^3$, HiWi and Adapter.

4.3 Gradient-based optimization

To effectively solve Equation (4), ApiQ utilizes a gradient-based algorithm akin to conventional neural network training. The process involves optimizing the quantized weight $\boldsymbol{Q}$ along with the low-rank matrices $\boldsymbol{A}$ and $\boldsymbol{B}$ jointly.

Originally, f from Equation (3) is a static function without any trainable parameters. Drawing inspiration from existing learnable quantization methods (Shao et al., 2023a; Liu et al., 2022b; Esser et al., 2020; Choi et al., 2018), ApiQ introduces two

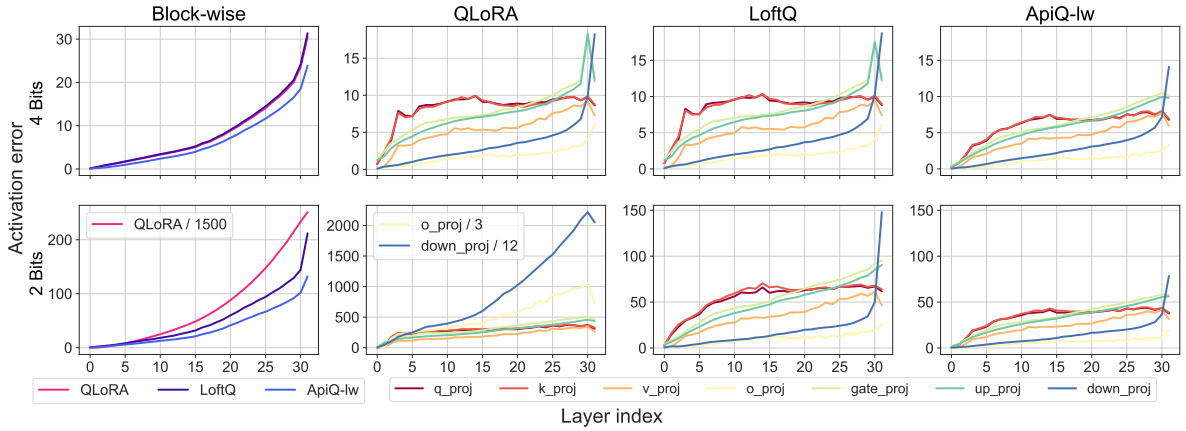

Figure 3: The average activation error $\|\boldsymbol{X}\boldsymbol{W} - \boldsymbol{X}^q(\boldsymbol{Q} + \boldsymbol{A}\boldsymbol{B}^\top)\|_F$ per token for different linear layers of Llama-2-7b. **1st column**: The activation error for every transformer block. We randomly sample 128 sentences from C4 to obtain the activations. For better visualization, some lines are divided by a factor, denoted as "/ factor". Please pay attention to the scale of the y-axis to compare different methods. ApiQ has the smallest activation error.

Algorithm 1 ApiQ-lw for one linear layer

Input: calibration samples $\boldsymbol{X}$ and $\boldsymbol{X}^q$, $\boldsymbol{W}$, $\boldsymbol{A}$, $\boldsymbol{B}$, Θ
Output: $\boldsymbol{Y}$, $\boldsymbol{Y}^q$, $\boldsymbol{A}$, $\boldsymbol{B}$, Θ
$\boldsymbol{Y} = \boldsymbol{X}\boldsymbol{W}$ ▷ Unquantized output, save for next layer
for $e = 0$ **to** (Epochs $- 1$) **do**
  for $(\boldsymbol{y}, \boldsymbol{x}^q)$ in $(\boldsymbol{Y}, \boldsymbol{X}^q)$ **do** ▷ Batch-wise
    $\boldsymbol{Q} = \boldsymbol{W}/s$
    $\boldsymbol{Q} = \lfloor \boldsymbol{Q} \rceil + \boldsymbol{Q} - \boldsymbol{Q}.\text{detach}()$ ▷ STE
    $\boldsymbol{Q} = s \cdot (\text{clamp}(\boldsymbol{Q} + z, 0, 2^b - 1) - z)$
    $\boldsymbol{y}^q = \boldsymbol{x}^q(\boldsymbol{Q} + \boldsymbol{A}\boldsymbol{B}^\top)$ ▷ Quantized output
    loss $= \text{MSE}(\boldsymbol{y} - \boldsymbol{y}^q)$
    loss.backward()
  end for
end for
$\boldsymbol{Y}^q = \boldsymbol{X}^q(\boldsymbol{Q} + \boldsymbol{A}\boldsymbol{B}^\top)$ ▷ Save for next layer

trainable parameters, γ and β, for each weight matrix. These parameters control the clipping range of the quantization process:

$$s = \frac{\sigma(\gamma)\max(\boldsymbol{W}) - \sigma(\beta)\min(\boldsymbol{W})}{2^b - 1}$$
$$z = -\lfloor \frac{\sigma(\beta)\min(\boldsymbol{W})}{s} \rceil$$

Here, σ denotes a sigmoid function, constraining the clipping range to prevent excessive value expansion. We initialize $\gamma = \beta = 4$ ($\sigma(4) \approx 0.98$) to maintain the original clipping range at the beginning of quantization.

The optimization for one linear layer is outlined in Algorithm 1. During this process, only $\boldsymbol{A}$, $\boldsymbol{B}$ and $\Theta = \{\gamma, \beta\}$ are trained. Given that the quantization function f incorporates a round-to-nearest operation, a straight-through estimator (STE) (Bengio et al., 2013) is applied to ensure the update of Θ. The ApiQ-lw algorithm is designed to be memory-efficient, optimizing each layer sequentially. This implies that any GPU capable of running the model inference can be used to quantize the model using ApiQ-lw. The outputs $\boldsymbol{Y}$ and $\boldsymbol{Y}^q$ from each layer serve as inputs to optimize the subsequent adjacent layer, ensuring efficient quantization.

ApiQ-bw employs a nearly identical optimization algorithm to ApiQ-lw, with the primary distinction being that the outputs, $\boldsymbol{y}$ and $\boldsymbol{y}^q$, are generated from a transformer block rather than a linear layer. It is worth noting that while ApiQ-bw offers improved time efficiency compared to ApiQ-lw, it necessitates marginally higher GPU memory usage due to the need to cache more activations from the layers within a transformer block.

Preliminary experiments. In Figure 3, ApiQ reduces the activation error by a large margin compared to QLoRA and LoftQ, more obvious for lower-bit quantization. Interestingly, while our objective is to minimize the activation error, the weight error of ApiQ is the smallest for most layers, as shown in Figure 2. This dual effectiveness in minimizing both activation and weight errors underscores the comprehensive nature of ApiQ to quantization. Further evidence of ApiQ's effectiveness is presented in Table 1 where ApiQ has the smallest performance gap for different trainable LoRA positions. In some cases, only training the LoRA modules in the attention position can offer the best results, similar to the original findings of LoRA (Hu et al., 2022). It suggests that ApiQ is particularly adept at addressing and mitigating the cumulative effects of quantization error.

Due to space constraints, we highly recommend that readers refer to Appendix §B.1 and §B.2 for

a comprehensive analysis of ApiQ's quantization quality and efficiency.

5 Experiments

In this section, we evaluate ApiQ on the language understanding, language modeling, arithmetic reasoning and commonsense reasoning tasks by quantizing DeBERTa-v3 (He et al., 2023), RoBERTa (Liu et al., 2019), Llama-2 (Touvron et al., 2023b) and Mistral (Jiang et al., 2023). Like QLoRA, LoftQ and LQ-LoRA, ApiQ consists of two steps: the quantization step and the finetuning step. During the quantization step, we initialize $\boldsymbol{Q}$, $\boldsymbol{A}$ and $\boldsymbol{B}$ in a way to preserve the starting point and mitigate the propagation of quantization error. For the finetuning step, we freeze $\boldsymbol{Q}$ in a lower bit and train $\boldsymbol{A}$ and $\boldsymbol{B}$ in half-precision (BFloat16).

Implementation details. In Algorithm 1, the quantization process of ApiQ requires a calibration dataset. We randomly sample 128 sentences from the training set of WikiText-2 (Merity et al., 2017). Following our baselines (Dettmers et al., 2023a; Li et al., 2023), LoRA modules are integrated into all linear layers. By default, the group/block size for quantization is 64 for all methods. We employ AdamW (Loshchilov and Hutter, 2019) as an optimizer to update $\boldsymbol{A}$, $\boldsymbol{B}$ and Θ. More implementation details for the quantization and finetuning steps are detailed in Appendix §C for reproduction.

Baselines include full finetuning (Full FT), LoRA (Hu et al., 2022), QLoRA (Dettmers et al., 2023a), GPTQ-LoRA (Frantar et al., 2022), LoftQ (Li et al., 2023), and LQ-LoRA (Guo et al., 2023). Full FT and LoRA are considered the upper bound for finetuning. QLoRA and GPTQ-LoRA employ NF-quantization and uniform quantization, respectively, on the pretrained weights with the default LoRA initialization. These methods are memory-efficient but distort the starting point. In contrast, LoftQ and LQ-LoRA initialize the matrices $\boldsymbol{Q}$, $\boldsymbol{A}$, and $\boldsymbol{B}$ to preserve the initial weight state, thus serving as a strong baseline to ApiQ.

5.1 Finetuning results and discussion

Natural language understanding. We finetune DeBERTa-v3-base and RoBERTa-large on the GLUE tasks (Wang et al., 2019) and show the results in Figure 4. ApiQ outperforms all baselines under the same level of quantization on average. With 3-bit quantization, ApiQ is even better or comparable to Full FT.

Language modeling. We finetune Llama-2-7b, Llama-2-13b and Mistral-7b-v0.1 on the WikiText-2 training set (Merity et al., 2017) and report their perplexity on the validation set, as shown in Figure 4 (Table C.7 for Mistral). Among the tested methods, ApiQ-bw consistently achieved the best performance, followed closely by ApiQ-lw across all bit levels. The performance difference becomes more pronounced at lower bit levels. The ApiQ's results on Llama-2-13b are even better than LoRA (Float16) for the 3-bit and 4-bit levels.

Arithmetic reasoning (single-task). We finetune Llama-2 and Mistral on the training set of GSM8K (Cobbe et al., 2021) and report the accuracy on the test set in Figure 4 (Table C.7 for Mistral). Similar to the results of WikiText-2, ApiQ-bw and ApiQ-lw achieve the highest and second-highest accuracy for all bit levels, respectively, with both ApiQs being comparable to or even better than LoRA for the 3 and 4-bit quantization.

ApiQ-lw or ApiQ-bw? ApiQ-lw is more memory-efficient for quantization than ApiQ-bw, as shown in Table B.3. However, ApiQ-lw requires more time for quantization due to the layer-by-layer manner. Based on the quantization quality (Appendix §B.1), quantization efficiency (Appendix §B.2) and the previously discussed finetuning results, we recommend using ApiQ-bw. Therefore, ApiQ-lw is ignored for the following experiments.

Arithmetic reasoning. The setting here contrasts with the previous experiments where each task involves finetuning a separate QLLM. Instead, we adopt a unified strategy by finetuning a single QLLM across all tasks as delineated in Hu et al. (2023). We finetune Llama-2 on Math10K (Hu et al., 2023), and evaluate the finetuned QLLM on the test sets of AQuA (Ling et al., 2017), GSM8K, MAWPS (Koncel-Kedziorski et al., 2016) and SVAMP (Patel et al., 2021). Such a setting is more practical as LLM is frequently used as a general model for various tasks.

As shown in Figure 4, ApiQ-bw consistently outperforms all quantization baselines for various bit levels, except for the 4-bit level where ApiQ is slightly worse than QLoRA, 53.5 vs. 53.7 for Llama-2-7b and 59.0 vs. 59.5 for Llama-2-13b. However, QLoRA's 3- and 2-bit results are extremely worse, $< 3\%$ accuracy (Table C.8).

Commonsense reasoning. In assessing the capacity of QLLM for commonsense reasoning, we focus on eight representative tasks: BoolQ (Clark et al., 2019), PIQA (Bisk et al., 2020),

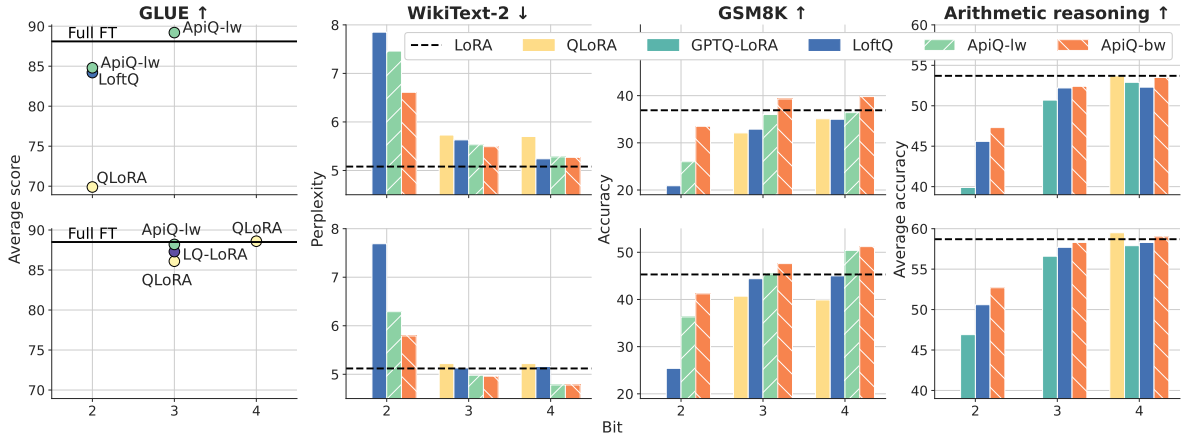

Figure 4: Finetuning performance over various tasks. **1st row**: LLM is DeBERTa-v3-base for GLUE and Llama-2-7b for the rest. **2nd row**: LLM is RoBERTa-large for GLUE and Llama-2-13b for the rest. For better visualization, some unexpectedly worse results are ignored. Please refer to Table C.4, C.7 and C.8 for the detailed numbers.

Model	Method	Bit	BoolQ	PIQA	SIQA	HellaS.	WinoG.	ARC-e	ARC-c	OBQA	Avg. ↑
Llama-2-7b	LoRA	16	$73.6_{0.3}$	$86.5_{0.1}$	$81.8_{0.1}$	$95.2_{0.1}$	$86.9_{0.2}$	$89.4_{0.4}$	$76.7_{0.8}$	$86.7_{0.8}$	$84.6_{0.2}$
	QLoRA	4	$\mathbf{73.9}_{0.4}$	$84.4_{0.8}$	$79.7_{0.2}$	$93.3_{0.2}$	$84.6_{0.6}$	$86.1_{0.4}$	$73.0_{0.4}$	$85.1_{0.5}$	$82.5_{0.2}$
	GPTQ-LoRA	4	$73.4_{0.4}$	$83.6_{0.5}$	$79.3_{0.3}$	$93.3_{0.1}$	$84.5_{0.8}$	$86.5_{0.3}$	$72.8_{1.3}$	$83.3_{0.2}$	$82.1_{0.2}$
	LoftQ	4	$73.7_{0.4}$	$86.0_{0.6}$	$81.1_{0.2}$	$94.6_{0.3}$	$86.3_{0.1}$	$88.1_{0.5}$	$75.5_{1.0}$	$\mathbf{86.2}_{0.6}$	$83.9_{0.2}$
	ApiQ-bw	4	$73.5_{0.2}$	$\mathbf{87.0}_{0.4}$	$\mathbf{82.0}_{0.1}$	$\mathbf{95.2}_{0.1}$	$\mathbf{86.9}_{0.2}$	$\mathbf{89.5}_{0.4}$	$\mathbf{77.0}_{0.8}$	$\mathbf{86.2}_{0.4}$	$\mathbf{84.7}_{0.2}$
	GPTQ-LoRA	3	$71.8_{0.2}$	$82.7_{0.3}$	$79.3_{0.6}$	$92.1_{0.1}$	$82.8_{0.3}$	$84.2_{0.6}$	$70.6_{0.8}$	$83.4_{1.1}$	$80.8_{0.1}$
	LoftQ	3	$\mathbf{74.0}_{0.0}$	$\mathbf{85.6}_{0.4}$	$81.0_{0.7}$	$94.3_{0.1}$	$85.6_{0.1}$	$\mathbf{88.1}_{0.6}$	$\mathbf{75.4}_{0.8}$	$85.5_{0.7}$	$83.7_{0.3}$
	ApiQ-bw	3	$73.3_{0.3}$	$\mathbf{85.6}_{0.1}$	$\mathbf{81.8}_{0.5}$	$\mathbf{94.6}_{0.0}$	$\mathbf{86.9}_{0.5}$	$87.9_{0.3}$	$73.7_{0.3}$	$\mathbf{86.4}_{1.3}$	$\mathbf{83.8}_{0.1}$
	GPTQ-LoRA	2	$62.2_{0.0}$	$49.5_{0.2}$	$33.3_{0.6}$	$25.1_{0.1}$	$49.4_{0.4}$	$25.0_{0.2}$	$22.6_{0.0}$	$27.6_{0.0}$	$36.8_{0.0}$
	LoftQ	2	$62.4_{0.0}$	$70.5_{2.9}$	$73.4_{0.5}$	$78.8_{3.6}$	$71.0_{3.7}$	$66.5_{4.5}$	$50.8_{4.3}$	$62.3_{7.5}$	$67.0_{3.3}$
	ApiQ-bw	2	$\mathbf{68.4}_{0.7}$	$\mathbf{80.7}_{0.3}$	$\mathbf{79.6}_{0.5}$	$\mathbf{91.4}_{0.1}$	$\mathbf{82.4}_{0.5}$	$\mathbf{82.7}_{0.8}$	$\mathbf{68.3}_{0.6}$	$\mathbf{80.5}_{0.6}$	$\mathbf{79.3}_{0.2}$
Llama-2-13b	LoRA	16	$76.3_{0.2}$	$88.5_{0.0}$	$83.4_{0.3}$	$96.5_{0.2}$	$89.6_{0.4}$	$92.8_{0.4}$	$81.7_{0.4}$	$89.6_{0.4}$	$87.3_{0.1}$
	QLoRA	4	$74.9_{0.5}$	$86.6_{0.5}$	$81.5_{0.5}$	$94.9_{0.1}$	$86.9_{0.2}$	$89.1_{0.7}$	$77.1_{0.4}$	$87.2_{0.7}$	$84.8_{0.3}$
	GPTQ-LoRA	4	$74.5_{0.6}$	$86.1_{1.0}$	$81.8_{0.2}$	$94.7_{0.3}$	$86.8_{0.1}$	$89.0_{0.1}$	$77.1_{0.9}$	$84.5_{1.2}$	$84.3_{0.0}$
	LoftQ	4	$76.0_{0.3}$	$87.9_{0.2}$	$82.8_{0.6}$	$95.8_{0.1}$	$88.9_{0.6}$	$91.2_{0.3}$	$80.8_{0.7}$	$88.8_{1.3}$	$86.5_{0.2}$
	ApiQ-bw	4	$\mathbf{76.2}_{0.3}$	$\mathbf{88.5}_{0.3}$	$\mathbf{83.5}_{0.1}$	$\mathbf{96.6}_{1.4}$	$\mathbf{90.0}_{0.4}$	$\mathbf{92.1}_{0.1}$	$\mathbf{81.2}_{0.3}$	$\mathbf{89.9}_{0.5}$	$\mathbf{87.3}_{0.1}$
	GPTQ-LoRA	3	$73.5_{0.5}$	$85.2_{0.3}$	$81.1_{0.5}$	$94.1_{0.1}$	$85.7_{0.3}$	$87.9_{0.4}$	$75.5_{0.7}$	$85.3_{0.9}$	$83.5_{0.0}$
	LoftQ	3	$75.2_{0.3}$	$87.8_{0.6}$	$\mathbf{82.8}_{0.2}$	$\mathbf{96.3}_{0.1}$	$\mathbf{89.5}_{0.4}$	$\mathbf{91.1}_{0.1}$	$\mathbf{81.4}_{0.5}$	$88.0_{0.5}$	$86.5_{0.2}$
	ApiQ-bw	3	$\mathbf{76.0}_{0.4}$	$\mathbf{88.0}_{0.5}$	$82.3_{0.1}$	$95.8_{0.0}$	$89.1_{0.1}$	$\mathbf{91.1}_{0.2}$	$81.1_{0.5}$	$\mathbf{89.5}_{0.4}$	$\mathbf{86.6}_{0.0}$
	GPTQ-LoRA	2	$62.2_{0.0}$	$50.1_{0.9}$	$34.0_{0.6}$	$25.1_{0.1}$	$49.6_{0.4}$	$25.0_{0.0}$	$22.7_{0.0}$	$27.6_{0.0}$	$37.1_{0.3}$
	LoftQ	2	$65.9_{0.1}$	$76.4_{0.3}$	$78.0_{0.5}$	$84.4_{0.7}$	$76.1_{0.4}$	$75.1_{0.1}$	$60.1_{0.4}$	$72.7_{1.4}$	$73.6_{0.2}$
	ApiQ-bw	2	$\mathbf{73.1}_{0.4}$	$\mathbf{85.2}_{0.5}$	$\mathbf{82.3}_{0.5}$	$\mathbf{94.4}_{0.1}$	$\mathbf{86.2}_{0.3}$	$\mathbf{88.2}_{0.3}$	$\mathbf{74.9}_{0.4}$	$\mathbf{85.9}_{1.4}$	$\mathbf{83.8}_{0.3}$

Table 2: Accuracy and standard deviation from three random runs on commonsense reasoning tasks.

SIQA (Sap et al., 2019), HellaSwag (Zellers et al., 2019), WinoGrande (Sakaguchi et al., 2020), ARC-e, ARC-c (Clark et al., 2018), and OBQA (Mihaylov et al., 2018). Similar to the multiple arithmetic reasoning tasks, we follow the setting of Hu et al. (2023), finetune a single QLLM on the combined training sets from these tasks, and report the accuracy of the test sets.

As shown in Table 2, ApiQ-bw consistently achieves the best average accuracy. For the 4-bit quantization, ApiQ-bw is the only method comparable to LoRA in Float16. For the 2-bit quantization, ApiQ-bw outperforms both GPTQ-LoRA and LoftQ by a large margin with an average accuracy improvement > 10%.

Finetune Llama-2-70b. Advised by a ARR reviewers, we conduct the finetuning experiments on Llama-2-70b to show the scalability of ApiQ, and show the results in Table 3. On average, ApiQ-bw still achieves the best performance.

Notably, the quantization procedure of Llama-2-70b with ApiQ-bw only uses 35GB memory and takes 11 hours, while LoftQ needs over 160GB memory and 7 hours. In addition, ApiQ utilizes the AutoGPTQ kernel[3], ensuring a true quantization setting. The peak GPU memory for finetuning is 78GB with a batch size of 2, without the need for gradient checkpointing or parameter/optimizer of-

[3]https://github.com/AutoGPTQ/AutoGPTQ

Method	Bit	GSM8K	SVAMP	MAWPS	AQuA	Avg. ↑
LoRA	16	73.6	80.7	89.1	31.9	68.8
QLoRA	2	0	0	0	0	0
GPTQ-LoRA	2	56.0	65.1	85.7	27.7	58.6
LoftQ	2	53.1	66.3	**89.5**	27.2	59.0
ApiQ-bw	2	**61.0**	**73.7**	88.7	**28.7**	**63.0**

Table 3: Accuracy on arithmetic reasoning tasks with Llama-2-70b.

Method	Bit	WikiText-2 ↓	C4 ↓	GSM8K	SVAMP	MAWPS	AQuA	Avg. ↑
Unquantized	16	6.14	8.88	-	-	-	-	-
LoRA	16	-	-	69.0	76.1	92.0	31.5	67.2
QLoRA	2	1e6	9e5	0.3	0.7	0.0	0.0	0.3
GPTQ-LoRA	2	NAN	NAN	31.6	50.4	82.8	**25.6**	47.6
LoftQ	2	7e4	4e4	34.0	53.4	**89.1**	21.7	49.5
ApiQ-bw	2	**13.07**	**21.37**	**48.7**	**66.6**	86.6	25.2	**56.8**

Table 4: Quantization quality and finetuning accuracy on Llama-3-8b.

floading. In contrast, LoftQ and QLoRA require at least two A100-80GB GPUs with parameter offloading (zero stage 3) and a batch size of 1, as they use the bitsandbytes kernel, which only supports 4-bit quantization.

Finetune Llama-3-8b. Also advised by a ARR reviewer that Llama-3 is more challenging to quantize (Huang et al., 2024), we further conduct the experiments on Llama-3-8b at the 2-bit level, and show the results in Table 4.

We do observe the difficulty of quantizing Llama-3-8b when comparing to the quantization results on Llama-2-7b in Table B.2. None of our baselines generate a reasonable perplexity on WikiText-2 and C4 with Llama-3-8b. However, ApiQ-bw can still have a much smaller perplexity. In addition, its finetuning performance is also much closer to the LoRA-finetuning performance on the FP16 model.

More results. We have discussed the most pertinent works in Section §3, and provide additional related works in Appendix §A due to space limitations. For further discussions, we recommend readers refer to (1) Appendix §B.3 for applying ApiQ to other PEFTs; (2) Appendix §B.4 for a discussion why ApiQ is effective for finetuning; and (3) Appendix §B.5 for ApiQ's sensitivity to the LoRA rank.

6 Conclusion

In this work, we propose ApiQ, a novel framework that aims to reduce the activation error during quantization by jointly quantizing the LLM's weights and initializing the LoRA's components. Extensive experiments, on five tasks across various encoder-only and decoder-only models, demonstrate ApiQ's effectiveness in adapting QLLM. It works extremely well with lower-bit quantization and larger models than the strong baselines. Further experiments also demonstrate ApiQ's capability as a pure PTQ method (Appendix §B.1).

Ethical Considerations

Finetuned QLLMs can improve accessibility tools, such as text-to-speech and translation services, benefiting individuals with disabilities and non-native language speakers. These models can also assist in educational settings by providing personalized tutoring and facilitating research through advanced data analysis capabilities. In healthcare, QLLMs can support professionals by synthesizing medical literature and aiding in patient communication, ultimately contributing to better healthcare outcomes.

However, the ability of QLLMs to generate human-like text raises concerns about the spread of misinformation. We advocate for the responsible deployment of these models, including mechanisms to detect and prevent the dissemination of false information. The powerful capabilities of QLLMs could be misused for malicious purposes, such as generating deceptive content or facilitating cyber-attacks. It is essential to develop robust security measures and ethical guidelines to prevent such misuse.

Limitations

Although ApiQ demonstrates impressive finetuning results, some limitations are inherited from its implementation. Compared to LoftQ (Li et al., 2023), ApiQ requires a calibration dataset to determine the clipping range of $\boldsymbol{W}$ and to initialize $\boldsymbol{A}$ and $\boldsymbol{B}$. This implementation has one obvious drawback: It requires more time for quantization, as shown in Table B.3. Since we only need to quantize the LLM once for finetuning various tasks, and the duration and GPU memory used for quantization are reasonable, we deem this limitation acceptable.

Secondly, we only evaluate ApiQ on a limited number of tasks with a total number of 5 models due to time and resource limitations. We couldn't guarantee its effectiveness on the other tasks and LLMs, and are still working on including more tasks and models, trying to show its generalization. For the post-training quantization results (Appendix §B.1), we didn't make sure ApiQ shares the same bit per parameter as our baselines, which makes the direct comparison unfair. Since the main focus of this research is about finetuning, we add these results mainly to raise attention to this new method for PTQ. In the future, we aim to apply ApiQ to quantize both weight and activation for faster inference.

Acknowledgements

We thank all reviewers of ICML2024 for their helpful feedback about block-wise ApiQ and more benchmarks, and the ARR reviews for their suggestions of more experiments on Llama-2-70b and Llama-3-8b. We also thank eBay Inc. for the computation support. This research was funded in part by the Netherlands Organization for Scientific Research (NWO) under project number VI.C.192.080.

References

Armen Aghajanyan, Sonal Gupta, and Luke Zettlemoyer. 2021. Intrinsic dimensionality explains the effectiveness of language model fine-tuning. In *Proceedings of the 59th Annual Meeting of the Association for Computational Linguistics and the 11th International Joint Conference on Natural Language Processing, ACL/IJCNLP 2021, (Volume 1: Long Papers), Virtual Event, August 1-6, 2021*, pages 7319–7328. Association for Computational Linguistics.

Yoshua Bengio, Nicholas Léonard, and Aaron C. Courville. 2013. Estimating or propagating gradients through stochastic neurons for conditional computation. *CoRR*, abs/1308.3432.

Yonatan Bisk, Rowan Zellers, Ronan Le Bras, Jianfeng Gao, and Yejin Choi. 2020. PIQA: reasoning about physical commonsense in natural language. In *The Thirty-Fourth AAAI Conference on Artificial Intelligence, AAAI 2020, The Thirty-Second Innovative Applications of Artificial Intelligence Conference, IAAI 2020, The Tenth AAAI Symposium on Educational Advances in Artificial Intelligence, EAAI 2020, New York, NY, USA, February 7-12, 2020*, pages 7432–7439. AAAI Press.

Tom B. Brown, Benjamin Mann, Nick Ryder, Melanie Subbiah, Jared Kaplan, Prafulla Dhariwal, Arvind Neelakantan, Pranav Shyam, Girish Sastry, Amanda Askell, Sandhini Agarwal, Ariel Herbert-Voss, Gretchen Krueger, Tom Henighan, Rewon Child, Aditya Ramesh, Daniel M. Ziegler, Jeffrey Wu, Clemens Winter, Christopher Hesse, Mark Chen, Eric Sigler, Mateusz Litwin, Scott Gray, Benjamin Chess, Jack Clark, Christopher Berner, Sam McCandlish, Alec Radford, Ilya Sutskever, and Dario Amodei. 2020. Language models are few-shot learners. *CoRR*, abs/2005.14165.

Tianqi Chen, Bing Xu, Chiyuan Zhang, and Carlos Guestrin. 2016. Training deep nets with sublinear memory cost. *CoRR*, abs/1604.06174.

Jungwook Choi, Zhuo Wang, Swagath Venkataramani, Pierce I-Jen Chuang, Vijayalakshmi Srinivasan, and Kailash Gopalakrishnan. 2018. PACT: parameterized clipping activation for quantized neural networks. *CoRR*, abs/1805.06085.

Christopher Clark, Kenton Lee, Ming-Wei Chang, Tom Kwiatkowski, Michael Collins, and Kristina Toutanova. 2019. Boolq: Exploring the surprising difficulty of natural yes/no questions. In *Proceedings of the 2019 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies, NAACL-HLT 2019, Minneapolis, MN, USA, June 2-7, 2019, Volume 1 (Long and Short Papers)*, pages 2924–2936. Association for Computational Linguistics.

Peter Clark, Isaac Cowhey, Oren Etzioni, Tushar Khot, Ashish Sabharwal, Carissa Schoenick, and Oyvind Tafjord. 2018. Think you have solved question answering? try arc, the AI2 reasoning challenge. *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.

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

Tim Dettmers, Artidoro Pagnoni, Ari Holtzman, and Luke Zettlemoyer. 2023a. Qlora: Efficient finetuning of quantized llms. *CoRR*, abs/2305.14314.

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

Jacob Devlin, Ming-Wei Chang, Kenton Lee, and Kristina Toutanova. 2019. BERT: pre-training of deep bidirectional transformers for language understanding. In *Proceedings of the 2019 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies, NAACL-HLT 2019, Minneapolis, MN, USA, June 2-7, 2019, Volume 1 (Long and Short Papers)*, pages 4171–4186. Association for Computational Linguistics.

Steven K. Esser, Jeffrey L. McKinstry, Deepika Bablani, Rathinakumar Appuswamy, and Dharmendra S. Modha. 2020. Learned step size quantization. In *8th International Conference on Learning Representations, ICLR 2020, Addis Ababa, Ethiopia, April 26-30, 2020*. OpenReview.net.

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

Aidan N. Gomez, Mengye Ren, Raquel Urtasun, and Roger B. Grosse. 2017. The reversible residual network: Backpropagation without storing activations. In *Advances in Neural Information Processing Systems 30: Annual Conference on Neural Information Processing Systems 2017, December 4-9, 2017, Long Beach, CA, USA*, pages 2214–2224.

Han Guo, Philip Greengard, Eric P. Xing, and Yoon Kim. 2023. Lq-lora: Low-rank plus quantized matrix decomposition for efficient language model finetuning. *CoRR*, abs/2311.12023.

Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. 2015. Delving deep into rectifiers: Surpassing human-level performance on imagenet classification. In *2015 IEEE International Conference on Computer Vision, ICCV 2015, Santiago, Chile, December 7-13, 2015*, pages 1026–1034. IEEE Computer Society.

Pengcheng He, Jianfeng Gao, and Weizhu Chen. 2023. Debertav3: Improving deberta using electra-style pre-training with gradient-disentangled embedding sharing. In *The Eleventh International Conference on Learning Representations, ICLR 2023, Kigali, Rwanda, May 1-5, 2023*. OpenReview.net.

Neil Houlsby, Andrei Giurgiu, Stanislaw Jastrzebski, Bruna Morrone, Quentin de Laroussilhe, Andrea Gesmundo, Mona Attariyan, and Sylvain Gelly. 2019. Parameter-efficient transfer learning for NLP. In *Proceedings of the 36th International Conference on Machine Learning, ICML 2019, 9-15 June 2019, Long Beach, California, USA*, volume 97 of *Proceedings of Machine Learning Research*, pages 2790–2799. PMLR.

Edward J. Hu, Yelong Shen, Phillip Wallis, Zeyuan Allen-Zhu, Yuanzhi Li, Shean Wang, Lu Wang, and Weizhu Chen. 2022. Lora: Low-rank adaptation of large language models. In *The Tenth International Conference on Learning Representations, ICLR 2022, Virtual Event, April 25-29, 2022*. OpenReview.net.

Zhiqiang Hu, Lei Wang, Yihuai Lan, Wanyu Xu, Ee-Peng Lim, Lidong Bing, Xing Xu, Soujanya Poria, and Roy Ka-Wei Lee. 2023. Llm-adapters: An adapter family for parameter-efficient fine-tuning of large language models. In *Proceedings of the 2023 Conference on Empirical Methods in Natural Language Processing, EMNLP 2023, Singapore, December 6-10, 2023*, pages 5254–5276. Association for Computational Linguistics.

Wei Huang, Xingyu Zheng, Xudong Ma, Haotong Qin, Chengtao Lv, Hong Chen, Jie Luo, Xiaojuan Qi, Xianglong Liu, and Michele Magno. 2024. An empirical study of llama3 quantization: From llms to mllms. *Preprint*, arXiv:2404.14047.

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.

Albert Q. Jiang, Alexandre Sablayrolles, Arthur Mensch, Chris Bamford, Devendra Singh Chaplot, Diego de Las Casas, Florian Bressand, Gianna Lengyel, Guillaume Lample, Lucile Saulnier, Lélio Renard Lavaud, Marie-Anne Lachaux, Pierre Stock, Teven Le Scao, Thibaut Lavril, Thomas Wang, Timothée Lacroix, and William El Sayed. 2023. Mistral 7b. *CoRR*, abs/2310.06825.

Albert Q. Jiang, Alexandre Sablayrolles, Antoine Roux, Arthur Mensch, Blanche Savary, Chris Bamford, Devendra Singh Chaplot, Diego de Las Casas, Emma Bou Hanna, Florian Bressand, Gianna Lengyel, Guillaume Bour, Guillaume Lample, Lélio Renard Lavaud, Lucile Saulnier, Marie-Anne Lachaux, Pierre Stock, Sandeep Subramanian, Sophia Yang, Szymon Antoniak, Teven Le Scao, Théophile Gervet, Thibaut Lavril, Thomas Wang, Timothée Lacroix, and William El Sayed. 2024. Mixtral of experts. *CoRR*, abs/2401.04088.

Diederik P. Kingma and Jimmy Ba. 2015. Adam: A method for stochastic optimization. In *3rd International Conference on Learning Representations, ICLR 2015, San Diego, CA, USA, May 7-9, 2015, Conference Track Proceedings*.

Rik Koncel-Kedziorski, Subhro Roy, Aida Amini, Nate Kushman, and Hannaneh Hajishirzi. 2016. MAWPS: A math word problem repository. In *NAACL HLT 2016, The 2016 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies, San Diego*

California, USA, June 12-17, 2016, pages 1152–1157. The Association for Computational Linguistics.

Yixiao Li, Yifan Yu, Chen Liang, Pengcheng He, Nikos Karampatziakis, Weizhu Chen, and Tuo Zhao. 2023. Loftq: Lora-fine-tuning-aware quantization for large language models. *CoRR*, abs/2310.08659.

Baohao Liao, Yan Meng, and Christof Monz. 2023a. Parameter-efficient fine-tuning without introducing new latency. In *Proceedings of the 61st Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers), ACL 2023, Toronto, Canada, July 9-14, 2023*, pages 4242–4260. Association for Computational Linguistics.

Baohao Liao, Shaomu Tan, and Christof Monz. 2023b. Make your pre-trained model reversible: From parameter to memory efficient fine-tuning. *CoRR*, abs/2306.00477.

Baohao Liao, David Thulke, Sanjika Hewavitharana, Hermann Ney, and Christof Monz. 2022. Mask more and mask later: Efficient pre-training of masked language models by disentangling the [MASK] token. In *Findings of the Association for Computational Linguistics: EMNLP 2022, Abu Dhabi, United Arab Emirates, December 7-11, 2022*, pages 1478–1492. Association for Computational Linguistics.

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.

Wang Ling, Dani Yogatama, Chris Dyer, and Phil Blunsom. 2017. Program induction by rationale generation: Learning to solve and explain algebraic word problems. In *Proceedings of the 55th Annual Meeting of the Association for Computational Linguistics, ACL 2017, Vancouver, Canada, July 30 - August 4, Volume 1: Long Papers*, pages 158–167. Association for Computational Linguistics.

Haokun Liu, Derek Tam, Mohammed Muqeeth, Jay Mohta, Tenghao Huang, Mohit Bansal, and Colin Raffel. 2022a. Few-shot parameter-efficient fine-tuning is better and cheaper than in-context learning. In *Advances in Neural Information Processing Systems 35: Annual Conference on Neural Information Processing Systems 2022, NeurIPS 2022, New Orleans, LA, USA, November 28 - December 9, 2022*.

Shih-Yang Liu, Chien-Yi Wang, Hongxu Yin, Pavlo Molchanov, Yu-Chiang Frank Wang, Kwang-Ting Cheng, and Min-Hung Chen. 2024. Dora: Weight-decomposed low-rank adaptation. *CoRR*, abs/2402.09353.

Yinhan Liu, Myle Ott, Naman Goyal, Jingfei Du, Mandar Joshi, Danqi Chen, Omer Levy, Mike Lewis, Luke Zettlemoyer, and Veselin Stoyanov. 2019. Roberta: A robustly optimized BERT pretraining approach. *CoRR*, abs/1907.11692.

Zechun Liu, Kwang-Ting Cheng, Dong Huang, Eric P. Xing, and Zhiqiang Shen. 2022b. Nonuniform-to-uniform quantization: Towards accurate quantization via generalized straight-through estimation. In *IEEE/CVF Conference on Computer Vision and Pattern Recognition, CVPR 2022, New Orleans, LA, USA, June 18-24, 2022*, pages 4932–4942. IEEE.

Ilya Loshchilov and Frank Hutter. 2019. Decoupled weight decay regularization. In *7th International Conference on Learning Representations, ICLR 2019, New Orleans, LA, USA, May 6-9, 2019*. OpenReview.net.

Stephen Merity, Caiming Xiong, James Bradbury, and Richard Socher. 2017. Pointer sentinel mixture models. In *5th International Conference on Learning Representations, ICLR 2017, Toulon, France, April 24-26, 2017, Conference Track Proceedings*. OpenReview.net.

Todor Mihaylov, Peter Clark, Tushar Khot, and Ashish Sabharwal. 2018. Can a suit of armor conduct electricity? A new dataset for open book question answering. In *Proceedings of the 2018 Conference on Empirical Methods in Natural Language Processing, Brussels, Belgium, October 31 - November 4, 2018*, pages 2381–2391. Association for Computational Linguistics.

Markus Nagel, Marios Fournarakis, Yelysei Bondarenko, and Tijmen Blankevoort. 2022. Overcoming oscillations in quantization-aware training. In *International Conference on Machine Learning, ICML 2022, 17-23 July 2022, Baltimore, Maryland, USA*, volume 162 of *Proceedings of Machine Learning Research*, pages 16318–16330. PMLR.

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

Adam Paszke, Sam Gross, Francisco Massa, Adam Lerer, James Bradbury, Gregory Chanan, Trevor Killeen, Zeming Lin, Natalia Gimelshein, Luca Antiga, Alban Desmaison, Andreas Köpf, Edward Z. Yang, Zachary DeVito, Martin Raison, Alykhan Tejani, Sasank Chilamkurthy, Benoit Steiner, Lu Fang, Junjie Bai, and Soumith Chintala. 2019. Pytorch: An imperative style, high-performance deep learning library. In *Advances in Neural Information Processing Systems 32: Annual Conference on Neural Information Processing Systems 2019, NeurIPS 2019, December 8-14, 2019, Vancouver, BC, Canada*, pages 8024–8035.

Arkil Patel, Satwik Bhattamishra, and Navin Goyal. 2021. Are NLP models really able to solve simple math word problems? In *Proceedings of the 2021 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies, NAACL-HLT 2021, Online, June 6-11, 2021*, pages 2080–2094. Association for Computational Linguistics.

Alec Radford, Jeffrey Wu, Rewon Child, David Luan, Dario Amodei, Ilya Sutskever, et al. 2019. Language

models are unsupervised multitask learners. *OpenAI blog*, 1(8):9.

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

Keisuke Sakaguchi, Ronan Le Bras, Chandra Bhagavatula, and Yejin Choi. 2020. Winogrande: An adversarial winograd schema challenge at scale. In *The Thirty-Fourth AAAI Conference on Artificial Intelligence, AAAI 2020, The Thirty-Second Innovative Applications of Artificial Intelligence Conference, IAAI 2020, The Tenth AAAI Symposium on Educational Advances in Artificial Intelligence, EAAI 2020, New York, NY, USA, February 7-12, 2020*, pages 8732–8740. AAAI Press.

Maarten Sap, Hannah Rashkin, Derek Chen, Ronan Le Bras, and Yejin Choi. 2019. Socialiqa: Commonsense reasoning about social interactions. *CoRR*, abs/1904.09728.

Teven Le Scao, Angela Fan, Christopher Akiki, Ellie Pavlick, Suzana Ilic, Daniel Hesslow, Roman Castagné, Alexandra Sasha Luccioni, François Yvon, Matthias Gallé, Jonathan Tow, Alexander M. Rush, Stella Biderman, Albert Webson, Pawan Sasanka Ammanamanchi, Thomas Wang, Benoît Sagot, Niklas Muennighoff, Albert Villanova del Moral, Olatunji Ruwase, Rachel Bawden, Stas Bekman, Angelina McMillan-Major, Iz Beltagy, Huu Nguyen, Lucile Saulnier, Samson Tan, Pedro Ortiz Suarez, Victor Sanh, Hugo Laurençon, Yacine Jernite, Julien Launay, Margaret Mitchell, Colin Raffel, Aaron Gokaslan, Adi Simhi, Aitor Soroa, Alham Fikri Aji, Amit Alfassy, Anna Rogers, Ariel Kreisberg Nitzav, Canwen Xu, Chenghao Mou, Chris Emezue, Christopher Klamm, Colin Leong, Daniel van Strien, David Ifeoluwa Adelani, and et al. 2022. BLOOM: A 176b-parameter open-access multilingual language model. *CoRR*, abs/2211.05100.

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

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

Shyam Anil Tailor, Javier Fernández-Marqués, and Nicholas Donald Lane. 2021. Degree-quant: Quantization-aware training for graph neural networks. In *9th International Conference on Learning Representations, ICLR 2021, Virtual Event, Austria, May 3-7, 2021*. OpenReview.net.

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. 2023a. Llama: Open and efficient foundation language models. *CoRR*, abs/2302.13971.

Hugo Touvron, Louis Martin, Kevin Stone, Peter Albert, Amjad Almahairi, Yasmine Babaei, Nikolay Bashlykov, Soumya Batra, Prajjwal Bhargava, Shruti Bhosale, Dan Bikel, Lukas Blecher, Cristian Canton-Ferrer, Moya Chen, Guillem Cucurull, David Esiobu, Jude Fernandes, Jeremy Fu, Wenyin Fu, Brian Fuller, Cynthia Gao, Vedanuj Goswami, Naman Goyal, Anthony Hartshorn, Saghar Hosseini, Rui Hou, Hakan Inan, Marcin Kardas, Viktor Kerkez, Madian Khabsa, Isabel Kloumann, Artem Korenev, Punit Singh Koura, Marie-Anne Lachaux, Thibaut Lavril, Jenya Lee, Diana Liskovich, Yinghai Lu, Yuning Mao, Xavier Martinet, Todor Mihaylov, Pushkar Mishra, Igor Molybog, Yixin Nie, Andrew Poulton, Jeremy Reizenstein, Rashi Rungta, Kalyan Saladi, Alan Schelten, Ruan Silva, Eric Michael Smith, Ranjan Subramanian, Xiaoqing Ellen Tan, Binh Tang, Ross Taylor, Adina Williams, Jian Xiang Kuan, Puxin Xu, Zheng Yan, Iliyan Zarov, Yuchen Zhang, Angela Fan, Melanie Kambadur, Sharan Narang, Aurélien Rodriguez, Robert Stojnic, Sergey Edunov, and Thomas Scialom. 2023b. Llama 2: Open foundation and fine-tuned chat models. *CoRR*, abs/2307.09288.

Mart van Baalen, Andrey Kuzmin, Markus Nagel, Peter Couperus, Cédric Bastoul, Eric Mahurin, Tijmen Blankevoort, and Paul N. Whatmough. 2024. GPTVQ: the blessing of dimensionality for LLM quantization. *CoRR*, abs/2402.15319.

Alex Wang, Amanpreet Singh, Julian Michael, Felix Hill, Omer Levy, and Samuel R. Bowman. 2019. GLUE: A multi-task benchmark and analysis platform for natural language understanding. In *7th International Conference on Learning Representations, ICLR 2019, New Orleans, LA, USA, May 6-9, 2019*. OpenReview.net.

Adina Williams, Nikita Nangia, and Samuel Bowman. 2018. A broad-coverage challenge corpus for sentence understanding through inference. In *Proceedings of the 2018 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies, Volume 1 (Long Papers)*, pages 1112–1122, New Orleans, Louisiana. Association for Computational Linguistics.

Thomas Wolf, Lysandre Debut, Victor Sanh, Julien Chaumond, Clement Delangue, Anthony Moi, Pierric Cistac, Tim Rault, Rémi Louf, Morgan Funtowicz, Joe Davison, Sam Shleifer, Patrick von Platen, Clara Ma, Yacine Jernite, Julien Plu, Canwen Xu, Teven Le Scao, Sylvain Gugger, Mariama Drame, Quentin Lhoest, and Alexander M. Rush. 2020. Transformers: State-of-the-art natural language processing. In *Proceedings of the 2020 Conference on Empirical Methods in Natural Language Processing: System*

Demonstrations, pages 38–45, Online. Association for Computational Linguistics.

Guangxuan Xiao, Ji Lin, Mickaël Seznec, Hao Wu, Julien Demouth, and Song Han. 2023. 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.

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.

Susan Zhang, Stephen Roller, Naman Goyal, Mikel Artetxe, Moya Chen, Shuohui Chen, Christopher Dewan, Mona T. Diab, Xian Li, Xi Victoria Lin, Todor Mihaylov, Myle Ott, Sam Shleifer, Kurt Shuster, Daniel Simig, Punit Singh Koura, Anjali Sridhar, Tianlu Wang, and Luke Zettlemoyer. 2022. OPT: open pre-trained transformer language models. *CoRR*, abs/2205.01068.

A Related Work

Large language models, trained on web-scale data for general tasks like masked word prediction (Liao et al., 2022; Devlin et al., 2019) or next-word prediction (Brown et al., 2020) in sentences, are crucial for transferring knowledge to various downstream tasks. These models have consistently achieved state-of-the-art results in a wide range of applications. Notably, scaling up LLMs has been observed to reliably improve performance in these downstream tasks. As a result, the size of LLMs has been steadily increasing, now reaching the remarkable scale of > 50 billion parameters (Jiang et al., 2024; Touvron et al., 2023b; Jiang et al., 2023; Zhang et al., 2022). In addition, instruction-finetuned LLMs (Jiang et al., 2023; Touvron et al., 2023b) reveal exceptional capabilities, such as enabling zero-shot or in-context learning (Radford et al., 2019; Brown et al., 2020).

Despite these advancements, transfer learning remains the predominant strategy for effectively applying these models to new task environments (Liu et al., 2022a; Brown et al., 2020). This approach, however, imposes unprecedented demands on computational resources, highlighting the need for efficient adaptation strategies. ApiQ reduces the GPU memory requirement for finetuning by loading the LLM's weights in a reduced bit format and reducing the number of trainable parameters. In addition, compared to QLoRA (Dettmers et al., 2023a) and its variants (Li et al., 2023; Guo et al., 2023), ApiQ demonstrates a very good manner for challenging lower-bit quantization, like 2 or 3 bits, which further reduces the GPU memory.

Post-training quantization (PTQ) converts high-precision LLM's weight values into discrete values for less memory usage. With the increasing size of LLMs, various PTQ methods (Xiao et al., 2023; Lin et al., 2023; Shao et al., 2023b; Dettmers et al., 2023b; Frantar et al., 2022) have been proposed to retain the full-precision LLM's performance while using less memory during inference. Notably, PTQ aims to maintain the performance of LLMs instead of adapting LLMs to new tasks. In addition, the performance of PTQ for lower bit-width degrades significantly. Even though ApiQ is a well-behaved PTQ method, its main purpose is to adapt LLMs to new tasks or retain full-precision LLM's performance for lower-bit quantization.

Quantization-aware training (QAT) is a technique where the model is trained to take into account the effects of quantization, typically reducing the precision of the model's parameters, to ensure minimal loss in performance when the model is deployed in a resource-constrained environment (Tailor et al., 2021; Nagel et al., 2022). Although QAT can be employed to adapt LLM to downstream tasks, it is memory-intensive because it involves quantization and full finetuning at the same time. In addition, some techniques, like straight-through estimator (Liu et al., 2022b), are required during full finetuning to calculate the gradients, being unstable for the training of LLM. In contrast, ApiQ separates the quantization and finetuning steps, making the finetuning stable, efficient and effective.

Method	Bit	Llama-2-7b WikiText ↓	Llama-2-7b C4 ↓	Llama-2-13b WikiText ↓	Llama-2-13b C4 ↓
-	16	5.47	6.97	4.88	6.46
QLoRA	4	5.65	7.16	4.98	6.57
LoftQ	4	5.62	7.16	4.96	<u>6.55</u>
ApiQ-lw	4	<u>5.55</u>	<u>7.08</u>	<u>4.95</u>	<u>6.55</u>
ApiQ-bw	4	**5.53**	**7.06**	**4.93**	**6.53**
QLoRA	3	1.8e5	2.4e5	9.6e4	1.2e5
LoftQ	3	10.72	12.79	6.89	8.72
ApiQ-lw	3	<u>5.87</u>	<u>7.58</u>	<u>5.18</u>	<u>6.88</u>
ApiQ-bw	3	**5.77**	**7.48**	**5.12**	**6.83**
QLoRA	2	1.8e5	2.4e5	9.7e4	1.3e5
LoftQ	2	1.0e3	6.7e2	59.94	72.64
ApiQ-lw	2	<u>16.25</u>	<u>23.93</u>	<u>10.89</u>	<u>15.83</u>
ApiQ-bw	2	**7.59**	**10.56**	**6.44**	**8.93**

Table B.1: The perplexity of ApiQ as a post-training quantization method without the finetuning step. The **best** and <u>second-best</u> results are in bold and underlined, respectively.

B More Results and Discussions

B.1 Quantization quality

In Section §4, we have demonstrated the superior quantization quality of ApiQ by comparing the weight and activation error after quantization. Here, we further evaluate ApiQ as a post-training quantization (PTQ) method, comparing it with other PTQ methods in a language modeling task.

We begin by comparing QLoRA (Dettmers et al., 2023a), LoftQ (Li et al., 2023) and ApiQ on the WikiText-2 test set (Merity et al., 2017) and the C4 validation set (Raffel et al., 2020), following the implementation details outlined in Appendix §C.2. For all methods, the quantization group size is set to 64, and the LoRA rank r is 64. As shown in Table B.1, ApiQ-bw and ApiQ-lw consistently achieve the best and second-best perplexity across various bit levels. Notably, the performance gap between ApiQ and the baselines widens at lower

Method	Bit	Group size	Llama-2-7b WikiText ↓	Llama-2-7b C4 ↓	Llama-2-13b WikiText ↓	Llama-2-13b C4 ↓
-	16	-	5.47	6.97	4.88	6.46
RTN	4	128	5.72	7.24	4.98	6.58
GPTQ	4	128	5.61	7.12	4.98	6.56
AWQ	4	128	5.62	7.13	4.97	6.56
OmniQuant	4	128	5.58	7.12	4.95	6.56
ApiQ-bw	4	128	**5.54**	**7.09**	**4.94**	**6.55**
RTN	3	128	6.66	8.40	5.51	7.18
GPTQ	3	128	6.29	7.89	5.42	7.00
AWQ	3	128	6.24	7.84	5.32	6.94
OmniQuant	3	128	6.03	7.75	5.28	6.98
ApiQ-bw	3	128	**5.86**	**7.63**	**5.20**	**6.92**
RTN	2	64	431.97	475.35	26.22	28.69
GPTQ	2	64	20.85	19.40	22.44	12.48
AWQ	2	64	2.1e5	1.6e5	1.2e5	9.5e4
OmniQuant	2	64	9.62	12.72	7.56	10.05
ApiQ-bw	2	64	**7.59**	**10.56**	**6.44**	**8.93**
RTN	2	128	4.2e3	4.9e3	122.08	139.65
GPTQ	2	128	36.77	33.70	28.14	20.97
AWQ	2	128	2.2e5	1.7e5	1.2e5	9.4e4
OmniQuant	2	128	11.06	15.02	8.26	11.05
ApiQ-bw	2	128	**8.25**	**12.04**	**6.71**	**9.13**

Table B.2: The comparison between ApiQ and other standard post-training quantization methods.

bit levels.

Next, we compare ApiQ to other standard PTQ methods such as round-to-nearest quantization (RTN), GPTQ (Frantar et al., 2022), AWQ (Lin et al., 2023), and OmniQuant (Shao et al., 2023a). We exclude ApiQ-lw in this comparison as ApiQ-bw demonstrates superior performance (refer to Table B.1). It is crucial to note that our objective is not to merely outperform existing PTQ methods. This is because the LoRA components in ApiQ are stored in FP16 format, inherently increasing the average bit-width per parameter, which makes direct comparisons with other PTQ methods less fair. Instead, our goal is to introduce a novel PTQ approach that mitigates quantization difficulty through the integration of LoRA components.

As illustrated in Table B.2, ApiQ-bw consistently delivers the smallest perplexity, with a more significant advantage at lower bit levels. ApiQ-bw can be viewed as a combination of OmniQuant and a new initialization of LoRA, as OmniQuant employs a similar quantization algorithm as Algorithm 1 without LoRA parameters. Nonetheless, ApiQ-bw outperforms OmniQuant, highlighting the effectiveness of jointly initializing the LoRA modules and quantizing the LLM weights.

A critical question arises: how does ApiQ compensate for the information loss inherent in quantization? The histograms of $\boldsymbol{Q}$, $\boldsymbol{A}$, and $\boldsymbol{B}$ in Figure B.1 provide insights into this process. Uniform quantization causes many values in $\boldsymbol{W}$ near the center to be mapped to the same value, leading to quantization error. ApiQ addresses this by centering $\boldsymbol{AB}^\top$ in this critical region. Additionally, the distribution span of ApiQ's $\boldsymbol{A}$ and $\boldsymbol{B}$ is significantly narrower compared to $\boldsymbol{W}$ and LoftQ, suggesting the potential for further quantizing $\boldsymbol{A}$ and $\boldsymbol{B}$ to reduce the overall bit-width per parameter.

Size	Method	Duration	Peak GPU memory
7b	GPTQ	0.2h	6GB
	OmniQuant	1.1h	12GB
	LoftQ	0.6h	14GB
	ApiQ-lw	4.1h	6GB
	ApiQ-bw	1.3h	12GB
13b	GPTQ	0.4h	9GB
	OmniQuant	2.2h	16GB
	LoftQ	1.3h	27GB
	ApiQ-lw	6.5h	9GB
	ApiQ-bw	2.4h	17GB

Table B.3: The duration and peak GPU memory used for quantizing Llama-2.

B.2 Quantization efficiency

In this section, we compare the duration and GPU memory usage of quantization between ApiQ and other baseline methods. Detailed implementation procedures for quantization are provided in Appendix §C.1. It is worth noting that an LLM needs

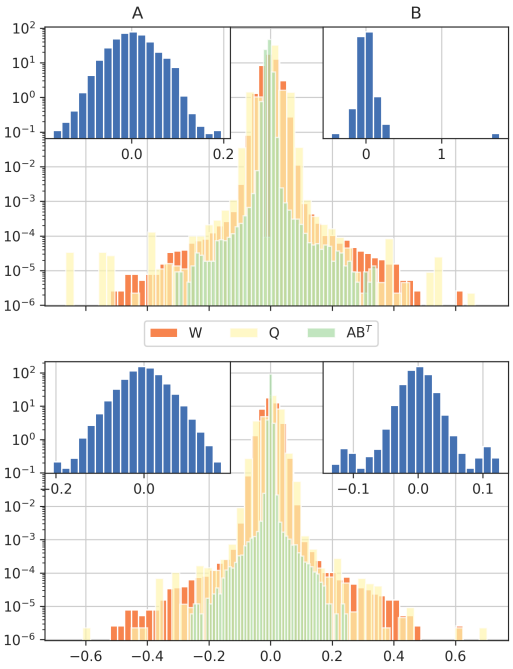

Figure B.1: Histogram of $\boldsymbol{Q}$, $\boldsymbol{A}$ and $\boldsymbol{B}$ for the 2-bit quantized output projection layer in the 30^{th} block of Llama-2-7b. **Upper**: LoftQ. **Lower**: ApiQ-lw. Refer to Figure C.2, C.3, C.4 and C.5 for all layers.

to be quantized only once and can then be saved for finetuning across various downstream tasks.

As shown in Table B.3, GPTQ stands out as the most efficient PTQ method, requiring the least time and GPU memory. ApiQ-lw uses a similar amount of GPU memory as GPTQ but requires more time due to its layer-by-layer sequential optimization. Similar to OmniQuant, ApiQ-bw consumes more memory than ApiQ-lw because it needs to cache more activations within a transformer block. However, ApiQ-bw is significantly more time-efficient than ApiQ-lw due to its block-by-block quantization approach. LoftQ requires the most GPU memory because of SVD. Overall, the resources required for ApiQ's quantization are reasonable and considerably lower than those needed for the finetuning step.

Based on the quantization quality, efficiency, and finetuning results (§5), we recommend using ApiQ-bw over ApiQ-lw.

B.3 ApiQ-bw for other PEFT

As discussed in Section §4.2, ApiQ-bw can easily be applied to other PEFT methods, because its block-by-block quantization manner is very friendly to them. Here we apply ApiQ-bw to a recent variant of LoRA, i.e. DoRA (Liu et al., 2024),

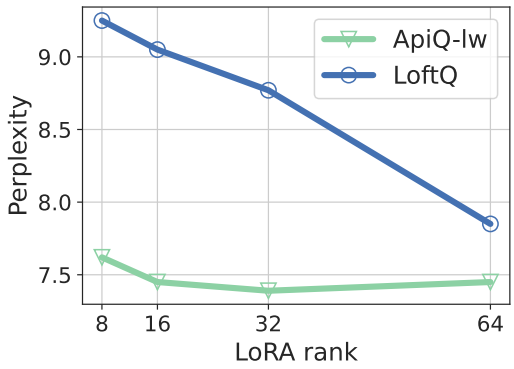

Figure B.2: Perplexity on WikiText-2 with 2-bit quantized Llama-2-7b for different LoRA ranks.

and show the finetuning results in Table B.4 and B.5. ApiQ-bw with DoRA outperforms QDoRA by a large margin, on average 76.2 vs. 36.6 for commonsense reasoning and 46.4 v.s. 1.4 for arithmetic reasoning.

B.4 Why ApiQ works so well

In this section, we discuss the reasons for the effectiveness of ApiQ.

The first reason is the smaller activation error of ApiQ. Compared to LoftQ and QLoRA, ApiQ's activation error is way more smaller. As shown in Table B.1, ApiQ has a much smaller perplexity. Maintaining a small activation error means that the learned knowledge from the full-precision LLM is preserved, thus facilitating the transfer learning for downstream tasks.

However, maintaining a smaller activation error is not the only reason for better finetuning results. Compared to LoftQ in Table B.1, GPTQ has a smaller perplexity in Table B.2. For example, GPTQ's perplexity is 20.85 for WikiText-2 on the 2-bit quantized Llama-2-7b, while LoftQ's perplexity is larger than 1000. Nevertheless, GPTQ-LoRA's [4] finetuning results are worse than LoftQ's, e.g. 39.9 vs. 45.6 for arithmetic reasoning and 36.8 vs. 67.0 for commonsense reasoning.

We hypothesize that the second reason is the better initialization of $\boldsymbol{A}$ and $\boldsymbol{B}$. The default initialization of $\boldsymbol{B} = \mathbf{0}$ is not friendly to training because of the constant value. $\boldsymbol{A}$ and $\boldsymbol{B}$ in ApiQ and LoftQ are initialized similarly to a Gaussian distribution (Figure B.1), which has been shown better for training (He et al., 2015).

[4] GPTQ-LoRA is LLM quantized by GPTQ and the LoRA's $\boldsymbol{B} = \mathbf{0}$.

Method	BoolQ	PIQA	SIQA	HellaS.	WinoG.	ARC-e	ARC-c	OBQA	Avg. ↑
QDoRA	62.2	49.7	33.2	24.4	48.8	24.4	22.8	27.2	36.6
ApiQ-bw with DoRA	68.7	78.8	76.9	85.5	79.8	78.5	62.8	78.4	76.2

Table B.4: ApiQ-bw with DoRA on the commonsense reasoning tasks with 2-bit quantized Llama-2-7b. QDoRA here means that we use QLoRA to quantize the LLM, initialize the LoRA module with default $\boldsymbol{B} = \mathbf{0}$, and train QLLM in the DoRA way, i.e. training the direction and magnitude separately. LoftQ can't be directly applied to DoRA, because DoRA has both addition and multiplication relation between the PEFT parameters and $\boldsymbol{W}$, and SVD can't be applied.

Method	GSM8K	SVAMP	MAWPS	AQuA	Avg. ↑
QDoRA	0.68	0.5	1.7	2.8	1.4
ApiQ-bw for DoRA	32.0	49.9	80.1	23.4	46.4

Table B.5: ApiQ-bw with DoRA on the arithmetic reasoning tasks with 2-bit quantized Llama-2-7b.

B.5 Performance vs. LoRA rank

In Figure B.2, we show the influence of LoRA rank for different methods. ApiQ is not sensitive to the LoRA rank, implying that ApiQ can be a more parameter-efficient finetuning method.

C Experimental Details

Like QLoRA (Dettmers et al., 2023a), LoftQ (Li et al., 2023) and LQ-LoRA (Guo et al., 2023), ApiQ consists of two steps: the quantization step and the finetuning step. During the quantization step, we initialize $\boldsymbol{Q}$, $\boldsymbol{A}$ and $\boldsymbol{B}$ in a way to preserve the starting point and mitigate the propagation of quantization error. For the finetuning step, we freeze $\boldsymbol{Q}$ in a lower bit and train $\boldsymbol{A}$ and $\boldsymbol{B}$ in half-precision (BFloat16). In this section, we describe the implementation details of these two steps for different tasks and LLMs. We run all experiments on NVIDIA A100-80GB or A6000-48GB with the training framework, Transformers (Wolf et al., 2020).

C.1 Quantization details for different LLMs

For all LLMs, 128 calibration sentences for the quantization step are randomly selected from the WikiText-2 training set (Merity et al., 2017). The hyper-parameters for the quantization step are detailed in Table C.1. By default, we incorporate the LoRA module into every linear layer.

DeBERTa and RoBERTa with ApiQ-lw. We apply ApiQ-lw to DeBERTa-v3-base (He et al., 2023) and RoBERTa-large (Liu et al., 2019), as these models are relatively small and efficiency concerns in quantization are minimal. Specifically, the duration for ApiQ-lw is 12 minutes for DeBERTa-v3-base and 1 hour for RoBERTa-large. The LoRA rank r is set to 32 for DeBERTa-v3-base, following Li et al. (2023), and 64 for RoBERTa-large, as per Guo et al. (2023). Given the relative simplicity of the GLUE tasks (Wang et al., 2019), we only employ 2-bit and 3-bit quantization. Unlike Li et al. (2023), we do not quantize the embedding layer and instead reproduce their experiments.

Llama-2 and Mistral with ApiQ-lw. We apply ApiQ-lw to Llama-2-7b, Llama-2-13b (Touvron et al., 2023b) and Mistral-7b-v0.1 (Jiang et al., 2023), with settings detailed in Table C.1.

Llama-2 and Mistral with ApiQ-bw. We also apply ApiQ-bw to Llama-2-7b, Llama-2-13b and Mistral-7b-v0.1, with the settings outlined in Table C.1. Compared to ApiQ-lw, we conduct an extensive search for the optimal learning rate and weight decay due to the time efficiency of ApiQ-bw. For instance, Llama-2-7b with ApiQ-lw requires 4 hours, whereas Llama-2-7b with ApiQ-bw requires only 1 hour.

To determine the best hyper-parameters, we evaluate the QLLM on the WikiText-2 test set (Merity et al., 2017) and the C4 validation set (Raffel et al., 2020), similar to the evaluation of post-training QLLM (see §C.2). The optimal hyper-parameter settings, determined by the lowest average perplexity across these two datasets, are listed for different LLMs in Table C.2.

C.2 Evaluation of QLLM

To assess the effectiveness of quantization, we adhere to the evaluation approach used in post-training quantization methods (van Baalen et al., 2024; Shao et al., 2023a; Xiao et al., 2023; Frantar et al., 2022; Dettmers et al., 2022). For the WikiText-2 test set (Merity et al., 2017), we apply

Hyper-parameter	DeBERTa & RoBERTa	Llama-2 & Mistral	Llama-2 & Mistral
ApiQ choice	ApiQ-lw	ApiQ-lw	ApiQ-bw
Optimizer	AdamW	AdamW	AdamW
Weight decay for Θ	0.1	0.1	{0, 0.001, 0.1}
Static LR for Θ	0.005	0.005	{0.001, 0.005, 0.01, 0.05}
Weight decay for $\boldsymbol{A}$ and $\boldsymbol{B}$	0.1	0.1	{0, 0.001, 0.1}
Static LR for $\boldsymbol{A}$ and $\boldsymbol{B}$	0.001	0.001	{0.0001, 0.0005, 0.001, 0.005}
Sequence length for calibration	128	1024	2048
Number of calibration samples	128	128	128
Epochs	20	20	20
Batch size	32	8	1
Group/block size for quantization	64	64	64
LoRA rank r	32 & 64	64	64

Table C.1: Hyper-parameter search space of the quantization step on different LLMs. Since ApiQ-bw is more time-efficient than ApiQ-lw, we conducted a more thorough search for ApiQ-bw. The best setting for ApiQ-bw is listed in Table C.2.

	Llama-2-7b			Llama-2-13b			Mistral-7b-v0.1		
Hyper-parameter	**4 Bits**	**3 Bits**	**2 Bits**	**4 Bits**	**3 Bits**	**2 Bits**	**4 Bits**	**3 Bits**	**2 Bits**
Weight decay for Θ	0.001	0.1	0.1	0.001	0.1	0.1	0.001	0.1	0.1
Static LR for Θ	0.05	0.001	0.005	0.01	0.001	0.005	0.01	0.001	0.005
Weight decay for $\boldsymbol{A}$ and $\boldsymbol{B}$	0.1	0	0.1	0	0	0.1	0	0	0.1
Static LR for $\boldsymbol{A}$ and $\boldsymbol{B}$	0.0001	0.0005	0.0005	0.0001	0.0005	0.0005	0.0001	0.0001	0.0005

Table C.2: Best hyper-parameter setting for different LLMs with ApiQ-bw. If one wants to apply ApiQ-bw to other LLMs, the settings from Llama-2-7b are universally well-performed and should be the first choice.

the QLLM to all sentences and calculate the average perplexity. For the validation set of C4 (Raffel et al., 2020), we use the "en/c4-validation.00000-of-00008.json.gz" split, concatenate all sentences, randomly cut off 256 sentences with a sequence length of 2048, and compute the average perplexity using the QLLM on these samples.

C.3 Natural language understanding

To study the language understanding ability of LLMs, we finetune quantized DeBERTa-v3-base (He et al., 2023) and RoBERTa-large (Liu et al., 2019) on the GLUE benchmark (Wang et al., 2019).

Finetuning details. The hyper-parameters for finetuning are outlined in Table C.3. We save the checkpoint every epoch, evaluate it on the development set, and report the best result. After deciding the best learning rate, three random runs are conducted and the median is reported in Table C.4.

C.4 Language modeling

To study whether the QLLM can preserve the language modeling ability after finetuning, we finetune quantized Llama-2-7b, Llama-2-13b and Mistral-7b-v0.1 on the WikiText-2 (Merity et al., 2017) training set and report the perplexity on the validation set.

Finetuning details. The hyper-parameters for finetuning are listed in Table C.5. We evaluate the finetuned QLLM on the validation set every epoch and report the best perplexity. After determining the best learning rate, we conduct three random runs and report the mean and standard deviation in Table C.7.

C.5 Arithmetic reasoning (single-task)

To study the arithmetic reasoning ability of QLLMs, we finetune quantized Llama-2-7b, Llama-2-13b and Mistral-7b-v0.1 on the GSM8K (Cobbe et al., 2021) training set and report the accuracy on the test set.

Finetuning details. The hyper-parameters for finetuning are listed in Table C.5. We evaluate the finetuned QLLM on the test set every epoch and report the best accuracy. After determining the best learning rate, we conduct three random runs and report the mean and standard deviation in Table C.7.

C.6 Arithmetic reasoning

The setting here contrasts with the previous experiments where each task involves finetuning a separate QLLM. Instead, we adopt a unified strategy by finetuning a single QLLM across all tasks

Hyper-parameter	RTE, MRPC, STS-B, CoLA	SST-2, QNLI, QQP, MNLI
Optimizer	AdamW	AdamW
Weight decay	0.1	0.1
LR	$\{0.1, 0.5, 1, 5\}\times10^{-4}$	$\{0.1, 0.5, 1, 5\}\times10^{-4}$
LR scheduler	Linear	Linear
Warmup ratio	10%	10%
Epochs	20	10
Batch size	32	32

Table C.3: Hyper-parameter search space for the finetuning on GLUE. For tasks with a number of training samples > 10K, we set the number of epochs as 10.

Model	Method	Bit	MNLI m/mm	QNLI Acc	QQP Acc/F1	SST-2 Acc	CoLA Matt	RTE Acc	MRPC Acc/F1	STS-B Pea/Spe	Avg. ↑
DeBERTa-base	Full FT*	16	90.5/90.6	94.0	92.4/89.8	95.3	69.2	82.0	89.5/93.3	91.6/91.1	88.1
	ApiQ-lw	3	90.3/90.2	93.9	92.6/90.1	95.8	71.9	85.9	91.7/94.0	91.5/91.3	89.2
	QLoRA*	2	79.9/79.5	83.7	88.6/84.7	86.9	N.A.	57.8	76.5/84.5	84.1/84.0	69.9
	LoftQ	2	88.5/88.5	92.7	91.6/88.7	94.7	63.6	64.6	88.5/91.8	89.2/89.0	84.2
	ApiQ-lw	2	88.4/88.7	92.3	91.7/89.0	94.6	64.2	67.1	89.5/92.4	90.2/89.9	84.8
RoBERTa-large	Full FT†	16	89.7	94.1	89.8	95.8	70.2	84.1	92.0	92.2	88.5
	QLoRA◇	4	-	-	-	-	-	-	-	-	88.6
	QLoRA†	3	89.8	94.3	89.9	96.4	64.3	70.8	92.0	91.6	86.1
	LQ-LoRA†	3	90.3	94.6	89.7	96.2	63.5	80.5	92.2	91.8	87.3
	ApiQ-lw	3	90.1/90.0	94.4	91.8/89.1	96.2	64.6	84.8	91.4/93.7	92.3/92.0	88.2

Table C.4: Results of encoder-only models on the GLUE development set. The LoRA rank r is 32 for DeBERTa-v3-base and 64 for RoBERTa-large. The median of three random runs is reported. We reproduce LoftQ because the published results are about quantizing both linear layers and the token embeddings. Here we only quantize the linear layers, keeping the same setting for all models in this paper. Results denoted by *, † and ◇ are from Li et al. (2023), Guo et al. (2023) and Dettmers et al. (2023a), respectively. When there is only one number for two metrics, it is an average over these two metric.

as delineated in Hu et al. (2023). We finetune Llama-2-7b and Llama-2-13b on Math10K (Hu et al., 2023) which is constructed from the training sets of GSM8K (Cobbe et al., 2021), MAWPS, MAWPS-single (Koncel-Kedziorski et al., 2016) and AQuA (Ling et al., 2017). Then we evaluate the finetuned QLLM on the test sets of AQuA, GSM8K, MAWPS and SVAMP (Patel et al., 2021). Such a setting is more practical as LLM is frequently used as a general model for various tasks.

Finetuning details. We follow Hu et al. (2023) to choose the hyper-parameters as in Table C.5. Instead of evaluating the finetuned QLLM every epoch, we only evaluate the trained model from the last epoch, simulating the practical finetuning scenario. We conduct three random runs and report the mean and standard deviation in Table C.8.

C.7 Commonsense reasoning

In assessing the capacity of QLLM for commonsense reasoning, we focus on eight representative tasks: BoolQ (Clark et al., 2019), PIQA (Bisk et al., 2020), SIQA (Sap et al., 2019), HellaSwag (Zellers et al., 2019), WinoGrande (Sakaguchi et al., 2020), ARC-e, ARC-c (Clark et al., 2018), and OBQA (Mihaylov et al., 2018). Similar to the multiple arithmetic reasoning tasks, we follow the setting of Hu et al. (2023) and finetune a single QLLM across all tasks. Specifically, the training and test sets from these eight tasks are reformulated according to a predefined template, so all tasks can be trained or evaluated in a generative way. Then we finetune Llama-2-7b and Llama-2-13b on the combined training set and report the accuracy on the test sets.

Finetuning details. We also borrow the finetuning recipe of Hu et al. (2023) as in Table C.5. We only evaluate the trained model from the last epoch (simulate a practical finetuning scenario), conduct three random runs, and report the mean and standard deviation in Table 2.

Hyper-parameter	WikiText-2	GSM8K	Arithmetic reasoning	Commonsense reasoning
Optimizer	AdamW		AdamW	
Weight decay	0.1		1.0	
LR	$\{0.1, 0.5, 0.7, 1, 3, 4\}\times10^{-4}$		3×10^{-4}	
LR scheduler	cosine		linear	
Warmup ratio	3%		10%	
Epochs	3	6	3	
Batch size	64	16	16	
Max sequence length	1024	512	512	

Table C.5: Hyper-parameter search space for the finetuning of Llama-2 and Mistral. Please refer to Table C.6 for the best learning rate for different LLMs on WikiText-2 and GSM8K.

Task	Method	Llama-2-7b			Llama-2-13b			Mistral-7b-v0.1		
		4 Bits	3 Bits	2 Bits	4 Bits	3 Bits	2 Bits	4 Bits	3 Bits	2 Bits
WikiText-2	ApiQ-lw	4e-4	3e-4	4e-4	3e-4	3e-4	3e-4	1e-4	7e-5	7e-5
	ApiQ-bw	3e-4	3e-4	3e-4	3e-4	3e-4	3e-4	1e-4	1e-4	1e-4
GSM8K	ApiQ-lw	3e-4	3e-4	3e-4	4e-4	4e-4	3e-4	7e-5	7e-5	7e-5
	ApiQ-bw	4e-4	4e-4	4e-4	3e-4	3e-4	4e-4	7e-5	7e-5	7e-5

Table C.6: Best learning rate for different LLMs on the WikiText-2 and GSM8K tasks.

Method	Bit	Llama-2-7b		Llama-2-13b		Mistral-7b-v0.1	
		WikiText (ppl↓)	GSM8K (acc↑)	WikiText (ppl↓)	GSM8K (acc↑)	WikiText (ppl↓)	GSM8K (acc↑)
LoRA	16	5.08	36.9	5.12	45.3	$5.17_{0.00}$	$52.2_{1.3}$
QLoRA	4	5.70	35.1	5.22	39.9	$\mathbf{5.25}_{0.00}$	$56.5_{1.1}$
LoftQ	4	**5.24**	35.0	5.16	45.0	$\mathbf{5.25}_{0.00}$	$56.7_{1.4}$
ApiQ-lw	4	$5.28_{0.00}$	$36.4_{0.5}$	$\mathbf{4.78}_{0.00}$	$50.4_{1.3}$	$5.32_{0.00}$	$57.2_{0.3}$
ApiQ-bw	4	$5.27_{0.00}$	$\mathbf{39.8}_{0.1}$	$\mathbf{4.78}_{0.00}$	$\mathbf{51.2}_{0.8}$	$5.26_{0.00}$	$\mathbf{59.2}_{0.1}$
QLoRA	3	5.73	32.1	5.22	40.7	$1540.26_{36.6}$	$50.5_{0.7}$
LoftQ	3	5.63	32.9	5.13	44.4	$6.82_{0.01}$	$51.6_{0.6}$
ApiQ-lw	3	$5.53_{0.01}$	$36.0_{0.3}$	$4.98_{0.00}$	$45.4_{1.1}$	$5.55_{0.00}$	$54.8_{1.7}$
ApiQ-bw	3	$\mathbf{5.49}_{0.00}$	$\mathbf{39.3}_{0.3}$	$\mathbf{4.96}_{0.00}$	$\mathbf{47.6}_{0.8}$	$\mathbf{5.48}_{0.00}$	$\mathbf{56.0}_{0.4}$
QLoRA	2	N.A.	N.A.	N.A.	N.A.	$1483.56_{12.2}$	$2.0_{0.3}$
LoftQ	2	7.85	20.9	7.69	25.4	$1849.32_{3.78}$	$1.7_{0.0}$
ApiQ-lw	2	$7.46_{0.00}$	$26.0_{0.4}$	$6.29_{0.00}$	$36.3_{0.5}$	$7.18_{0.00}$	$41.3_{0.8}$
ApiQ-bw	2	$\mathbf{6.61}_{0.00}$	$\mathbf{33.5}_{0.5}$	$\mathbf{5.79}_{0.00}$	$\mathbf{41.2}_{0.9}$	$\mathbf{6.69}_{0.00}$	$\mathbf{45.0}_{0.1}$

Table C.7: Finetuning results of WikiText and GSM8K on Llama-2-7b, Llama-2-13b and Mistral-7b-v0.1. Results without standard deviation are from Li et al. (2023).

Method	Bit	Llama-2-7b					Llama-2-13b				
		GSM8K	SVAMP	MAWPS	AQuA	Avg. ↑	GSM8K	SVAMP	MAWPS	AQuA	Avg. ↑
LoRA	16	$43.6_{0.7}$	$59.4_{1.7}$	$85.0_{1.7}$	$27.0_{2.0}$	$53.7_{0.6}$	$55.3_{0.5}$	$67.7_{0.9}$	$87.4_{0.7}$	$24.4_{0.9}$	$58.7_{0.2}$
QLoRA	4	$42.7_{0.4}$	$58.7_{0.7}$	$\mathbf{87.3}_{1.9}$	$\mathbf{26.4}_{1.6}$	$\mathbf{53.7}_{0.6}$	$54.8_{0.5}$	$\mathbf{69.4}_{0.3}$	$87.0_{0.7}$	$\mathbf{26.8}_{1.0}$	$\mathbf{59.5}_{0.3}$
GPTQ-LoRA	4	$43.0_{0.9}$	$58.4_{0.6}$	$86.1_{0.7}$	$24.3_{0.8}$	$52.9_{0.3}$	$53.2_{0.9}$	$67.5_{1.2}$	$85.3_{0.7}$	$25.6_{2.6}$	$57.9_{1.0}$
LoftQ	4	$41.7_{0.6}$	$56.0_{0.8}$	$86.3_{0.5}$	$25.3_{1.0}$	$52.3_{0.5}$	$54.9_{1.4}$	$66.5_{0.7}$	$87.7_{0.5}$	$23.9_{1.6}$	$58.3_{0.6}$
ApiQ-bw	4	$\mathbf{43.2}_{0.9}$	$\mathbf{59.0}_{0.9}$	$85.7_{0.7}$	$26.0_{1.8}$	$53.5_{0.8}$	$\mathbf{55.3}_{0.6}$	$67.4_{0.5}$	$\mathbf{87.8}_{0.9}$	$25.6_{0.2}$	$59.0_{0.4}$
QLoRA	3	$1.4_{0.2}$	$1.4_{0.3}$	$0.7_{0.5}$	$3.4_{1.5}$	$1.7_{0.5}$	$0.8_{0.6}$	$2.5_{2.2}$	$0.3_{0.2}$	$6.2_{6.8}$	$2.4_{2.0}$
GPTQ-LoRA	3	$38.9_{0.4}$	$55.7_{1.2}$	$84.9_{0.3}$	$23.2_{1.6}$	$50.7_{0.9}$	$50.6_{0.0}$	$65.2_{1.5}$	$88.0_{1.0}$	$22.6_{1.3}$	$56.6_{0.8}$
LoftQ	3	$39.9_{0.4}$	$\mathbf{56.3}_{2.2}$	$86.3_{0.8}$	$\mathbf{26.4}_{1.4}$	$52.2_{0.7}$	$\mathbf{53.9}_{1.2}$	$66.1_{0.2}$	$87.0_{0.9}$	$23.6_{0.7}$	$57.7_{0.3}$
ApiQ-bw	3	$\mathbf{41.4}_{1.5}$	$55.9_{0.3}$	$\mathbf{87.0}_{1.4}$	$25.2_{0.9}$	$\mathbf{52.4}_{0.6}$	$51.5_{0.8}$	$\mathbf{67.4}_{0.3}$	$\mathbf{88.5}_{1.2}$	$\mathbf{25.6}_{1.3}$	$\mathbf{58.3}_{0.3}$
QLoRA	2	$0.9_{0.4}$	$1.5_{1.1}$	$0.8_{0.7}$	$5.1_{4.9}$	$2.1_{1.7}$	$0.5_{0.4}$	$0.7_{0.9}$	$0.1_{0.2}$	$0.9_{1.3}$	$0.6_{0.4}$
GPTQ-LoRA	2	$21.7_{0.6}$	$39.0_{1.3}$	$76.6_{0.8}$	$22.1_{1.8}$	$39.9_{0.5}$	$31.9_{0.0}$	$49.6_{1.0}$	$82.5_{0.4}$	$\mathbf{23.6}_{0.9}$	$46.9_{0.5}$
LoftQ	2	$29.5_{0.8}$	$45.8_{0.7}$	$\mathbf{83.6}_{0.6}$	$23.2_{2.0}$	$45.6_{0.7}$	$37.0_{0.6}$	$55.9_{0.8}$	$\mathbf{87.7}_{1.3}$	$21.7_{1.1}$	$50.6_{0.2}$
ApiQ-bw	2	$\mathbf{31.2}_{0.5}$	$\mathbf{51.0}_{1.1}$	$82.9_{1.6}$	$\mathbf{23.9}_{1.0}$	$\mathbf{47.3}_{0.5}$	$\mathbf{43.1}_{0.8}$	$\mathbf{59.2}_{1.2}$	$85.1_{1.1}$	$23.4_{1.4}$	$\mathbf{52.7}_{0.5}$

Table C.8: Accuracy on four arithmetic reasoning tasks. All methods use the same hyper-parameters as listed in Table C.5. The LoRA rank r is 64 for all methods.

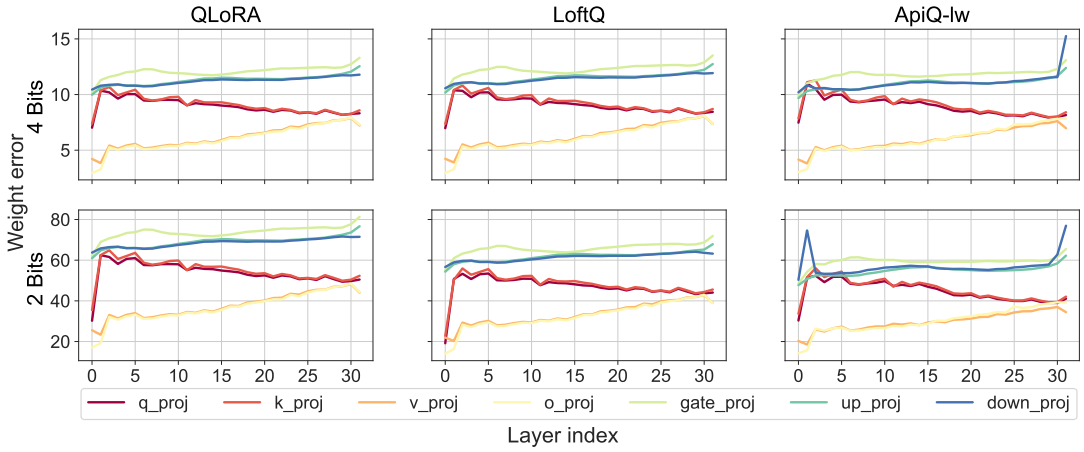

Figure C.1: The weight quantization error $||\boldsymbol{W} - (\boldsymbol{Q} + \boldsymbol{A}\boldsymbol{B}^T)||_F$ for different linear layers of Llama-2-7B. For 4-bit quantization, all methods are comparable, because 4-bit quantization doesn't significantly break down the starting point. For 2-bit quantization, ApiQ has the smallest quantization error for most layers, although its goal is to minimize the activation error.

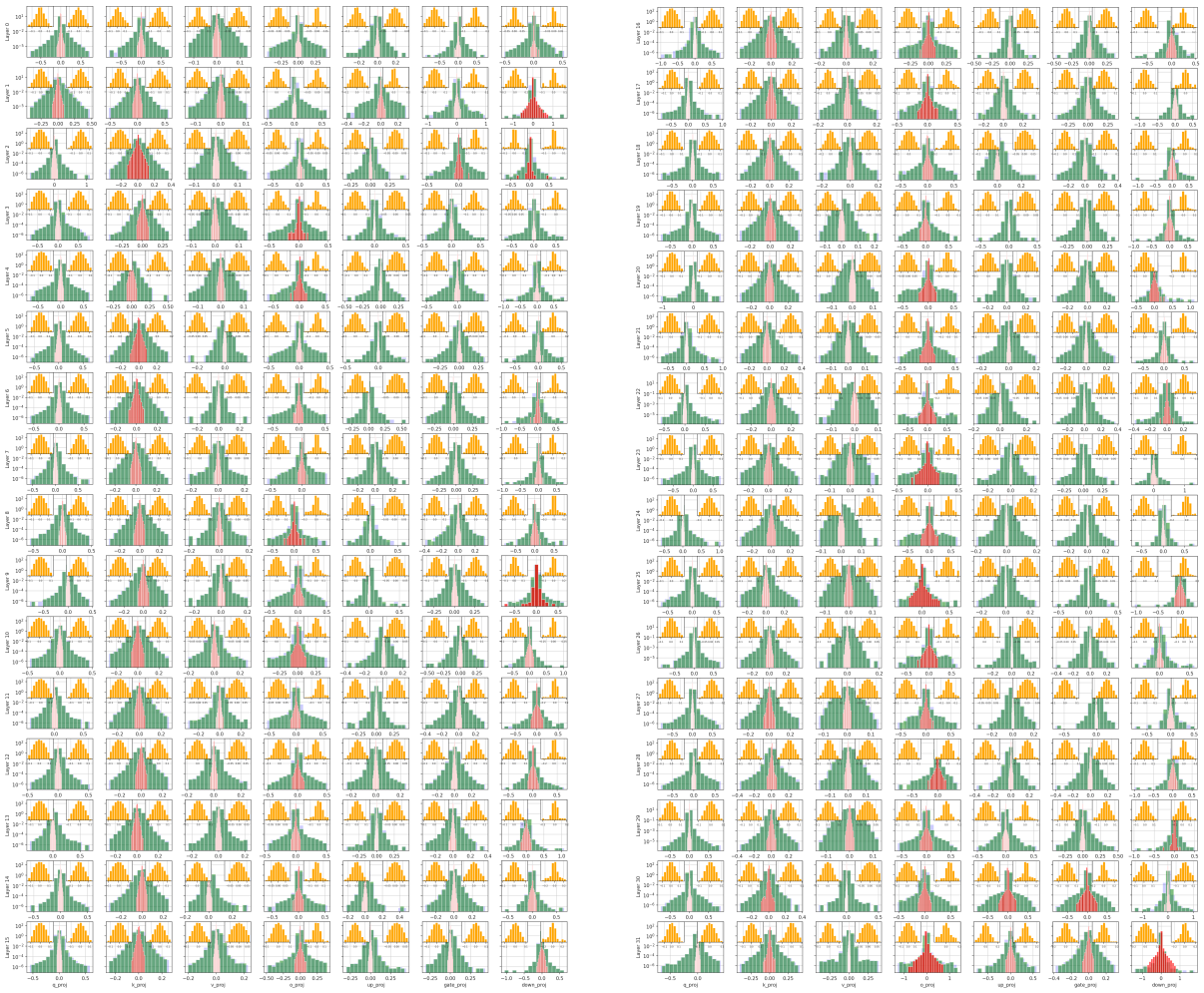

Figure C.2: Histogram of $\boldsymbol{Q}$, $\boldsymbol{A}$ and $\boldsymbol{B}$ for the **4-bit** quantized layer of Llama-2-7b with **ApiQ-lw**. Blue: $\boldsymbol{W}$. Green: $\boldsymbol{Q}$. Red: $\boldsymbol{A}\boldsymbol{B}^\top$. Orange: $\boldsymbol{A}$(Left) or $\boldsymbol{B}$(Right). Compared to LoftQ, the distribution of $\boldsymbol{B}$ of ApiQ is symmetric and doesn't have outliers, which might be one reason why ApiQ outperforms LoftQ.

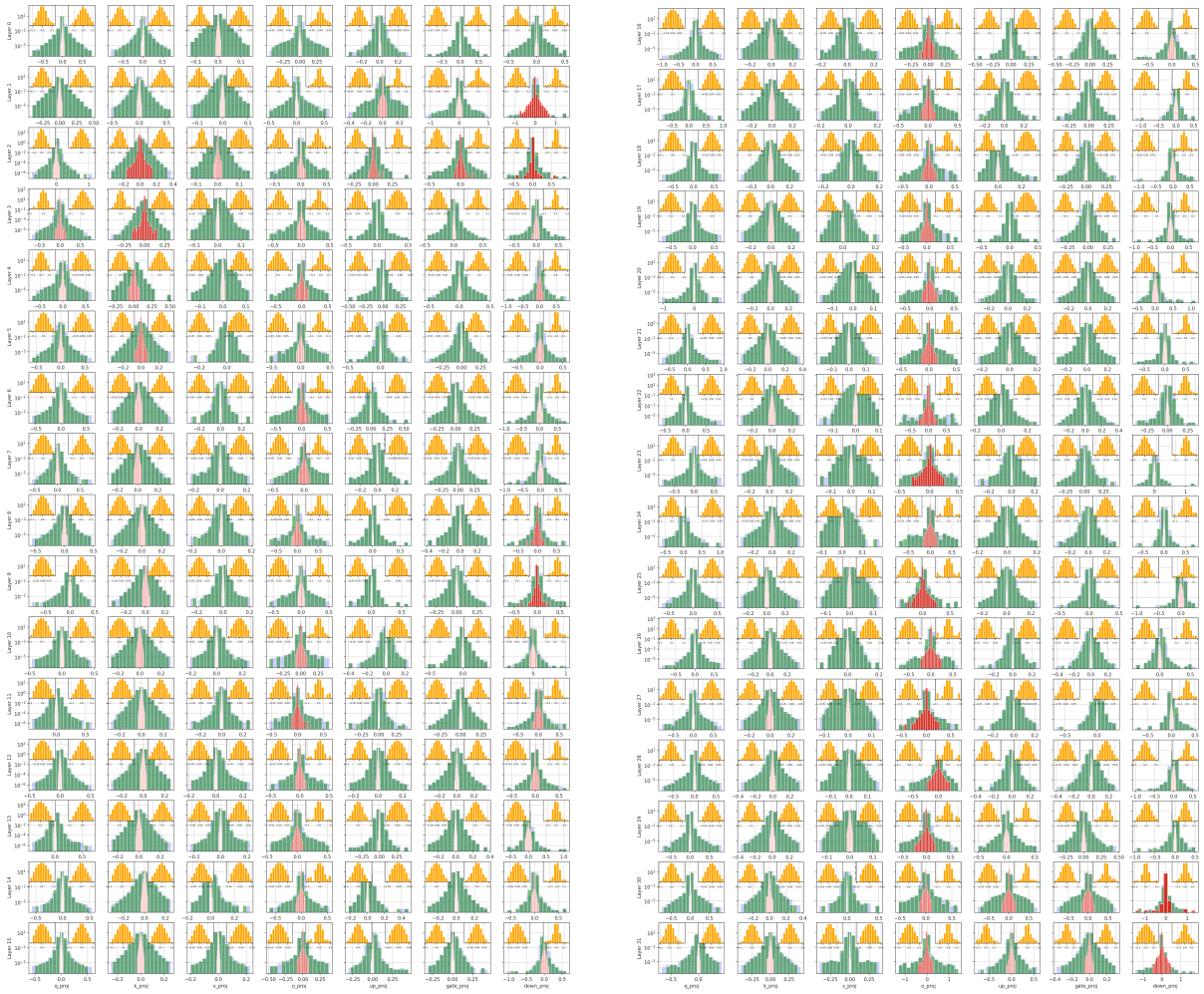

Figure C.3: Histogram of $\boldsymbol{Q}$, $\boldsymbol{A}$ and $\boldsymbol{B}$ for the **3-bit** quantized layer of Llama-2-7b with **ApiQ-lw**. Blue: $\boldsymbol{W}$. Green: $\boldsymbol{Q}$. Red: $\boldsymbol{AB}^\top$. Orange: $\boldsymbol{A}$(Left) or $\boldsymbol{B}$(Right). Compared to LoftQ, the distribution of $\boldsymbol{B}$ of ApiQ is symmetric and doesn't have outliers, which might be one reason why ApiQ outperforms LoftQ.

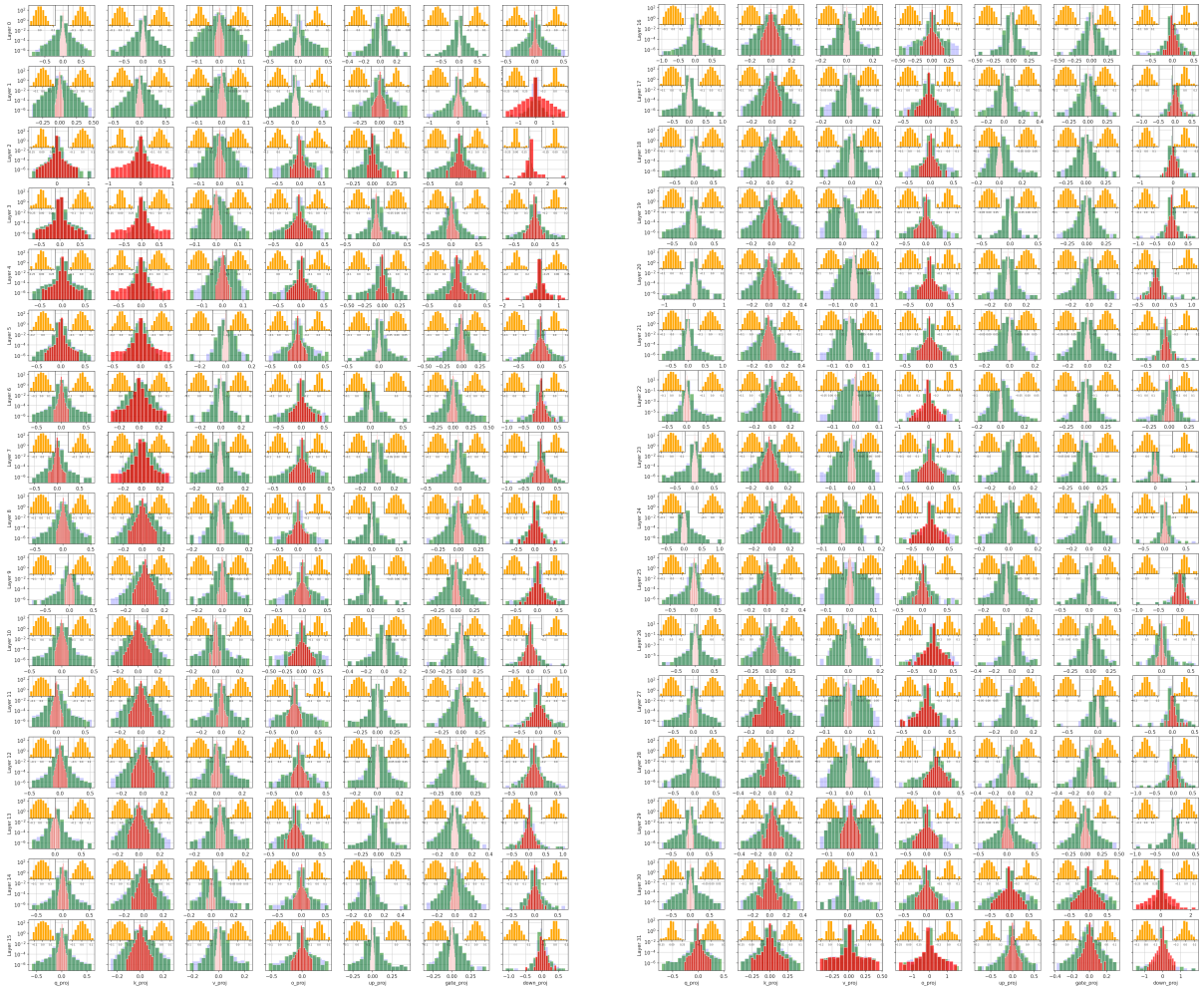

Figure C.4: Histogram of $\boldsymbol{Q}$, $\boldsymbol{A}$ and $\boldsymbol{B}$ for the **2-bit** quantized layer of Llama-2-7b with **ApiQ-lw**. Blue: $\boldsymbol{W}$. Green: $\boldsymbol{Q}$. Red: $\boldsymbol{AB}^\top$. Orange: $\boldsymbol{A}$(Left) or $\boldsymbol{B}$(Right). Compared to LoftQ, the distribution of $\boldsymbol{B}$ of ApiQ is symmetric and doesn't have outliers, which might be one reason why ApiQ outperforms LoftQ.

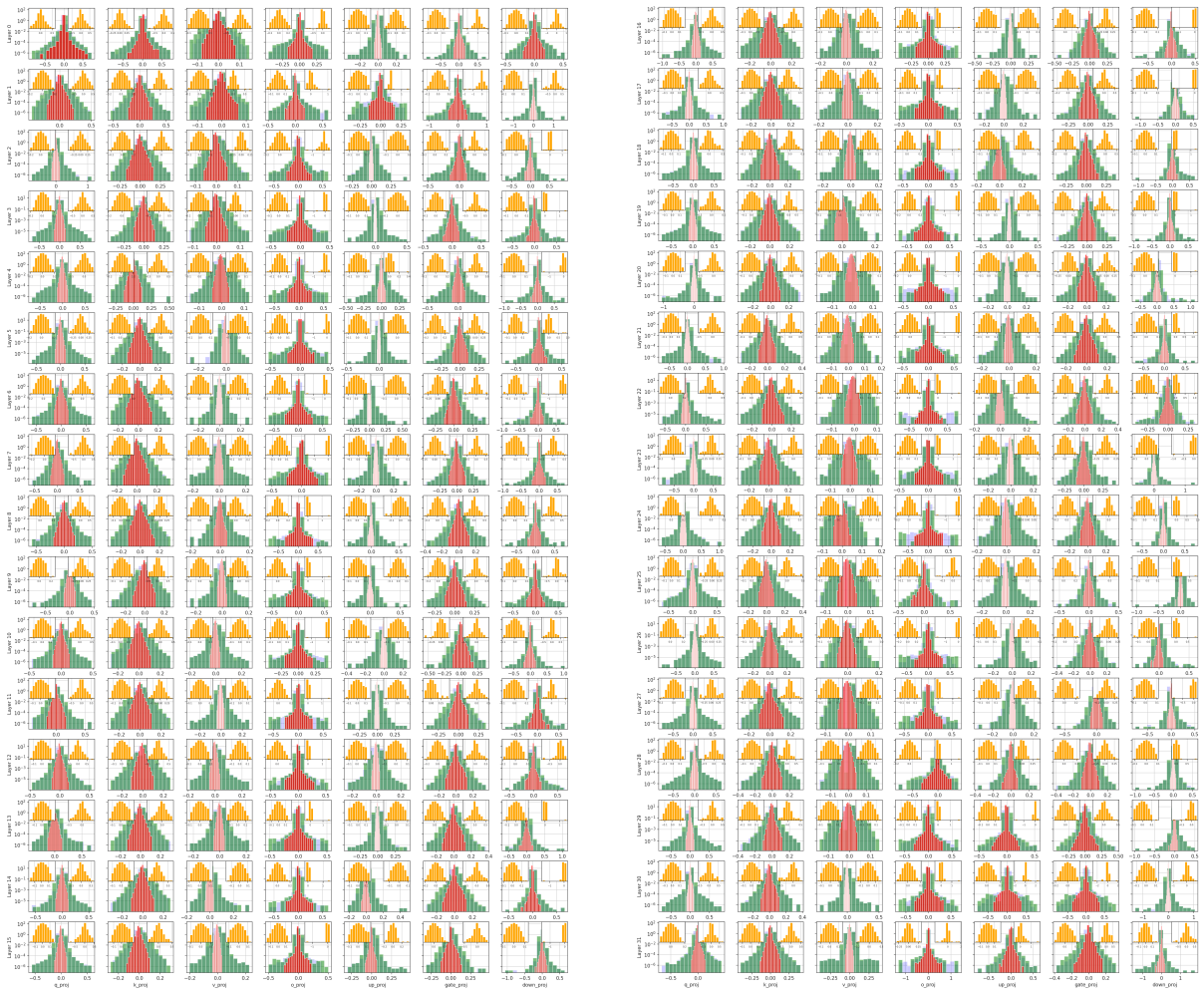

Figure C.5: Histogram of $\boldsymbol{Q}$, $\boldsymbol{A}$ and $\boldsymbol{B}$ for the **2-bit** quantized layer of Llama-2-7b with **LoftQ**. Blue: $\boldsymbol{W}$. Green: $\boldsymbol{Q}$. Red: $\boldsymbol{AB}^\top$. Orange: $\boldsymbol{A}$(Left) or $\boldsymbol{B}$(Right). Compared to ApiQ, the distribution of $\boldsymbol{B}$ of LoftQ is asymmetric for most linear layers and has many outliers, which might be one reason why LoftQ performs worse for 2-bit quantization.