QEFT: Quantization for Efficient Fine-Tuning of LLMs

Changhun Lee[1*] **Jungyu Jin**[2*] **Younghyun Cho**[2] **Eunhyeok Park**[2]
[1]Department of Convergence IT Engineering
[2]Graduate School of Artificial Intelligence
Pohang University of Science and Technology (POSTECH)
{changhun.lee, jgjin0317, yhcho97, eh.park}@postech.ac.kr

Abstract

With the rapid growth in the use of fine-tuning for large language models (LLMs), optimizing fine-tuning while keeping inference efficient has become highly important. However, this is a challenging task as it requires improvements in all aspects, including inference speed, fine-tuning speed, memory consumption, and, most importantly, model quality. Previous studies have attempted to achieve this by combining quantization with fine-tuning, but they have failed to enhance all four aspects simultaneously. In this study, we propose a new lightweight technique called Quantization for Efficient Fine-Tuning (QEFT). QEFT accelerates both inference and fine-tuning, is supported by robust theoretical foundations, offers high flexibility, and maintains good hardware compatibility. Our extensive experiments demonstrate that QEFT matches the quality and versatility of full-precision parameter-efficient fine-tuning, while using fewer resources. Our code is available at https://github.com/xvyaward/qeft.

1 Introduction

While the outstanding zero-shot performance of large language models (LLMs) (Brown et al., 2020; Radford et al., 2019; Touvron et al., 2023; Zhang et al., 2022) significantly contributes to their popularity, their versatility and adaptability are also crucial factors in their widespread adoption. Through transfer learning and fine-tuning, LLMs can be extended to handle unseen or complex tasks, including new data types, which opens up new possibilities across various applications (Qin et al., 2023; Hao et al., 2024). As efficient inference and fine-tuning become increasingly important, this paper explores methods to enhance the efficiency of both inference and fine-tuning for LLMs.

Examining past research, several methods have been proposed to enhance the efficiency of inference, including pruning (Frantar and Alistarh, 2023; Sun et al., 2023), speculative decoding (Leviathan et al., 2023; Miao et al., 2023), KV caching (Hooper et al., 2024), and, particularly, weight quantization (Frantar et al., 2022; Lin et al., 2023; Yao et al., 2022; Shao et al., 2023; Lee et al., 2024). However, studies focusing on lightweight approaches for fine-tuning remain relatively limited. This is because when considering both fine-tuning and inference, the factors requiring optimization—such as inference speed, training speed, memory consumption, and accuracy—become significantly varied. Balancing all these conditions simultaneously presents a substantial challenge.

In this context, LoRA (Hu et al., 2022) is a representative study enabling parameter-efficient fine-tuning (PEFT) (Houlsby et al., 2019; Zaken et al., 2022; Liu et al., 2022; Hu et al., 2022) by freezing the pre-trained weights while adding a decomposed path that undergoes updates. This approach opens up new opportunities for updating LLMs flexibly with limited resources, leading to numerous emerging applications (Kim et al., 2023; Lee et al., 2023; Wang et al., 2023). In addition, several studies have advanced this concept by attempting to harmonize weight quantization and LoRA to reap the benefits of both methods. For instance, in QLoRA (Dettmers et al., 2023) and QA-LoRA (Xu

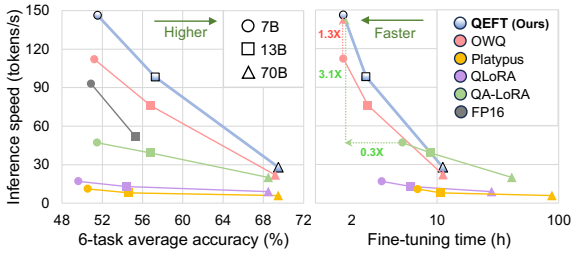


Figure 1: Pareto-front comparison of PEFT methods. (Left): Average few-shot accuracy after fine-tuning vs. inference speed. (Right): Fine-tuning time vs. inference speed. For more details, please see Sec. 6.1.

*These authors contributed equally.

Findings of the Association for Computational Linguistics: EACL 2024, pages 13823–13837
November 12-16, 2024

et al., 2023), the pre-trained weights remain fixed after quantization, while only the FP16 low-rank path is added and updated exclusively. However, QLoRA exhibits slow inference speeds, and both methods still entail noticeable fine-tuning overhead. Achieving improvements in all aspects of inference and fine-tuning is not easily accomplished by simply applying multiple optimizations in parallel.

In this study, we propose a new quantization technique called Quantization for Efficient Fine-Tuning (QEFT), designed to achieve optimal performance in both inference and training. This method employs the data format of OWQ (Lee et al., 2024), storing weak columns vulnerable to quantization in FP16 while storing the majority of weights in 4-bit or less, and updating only the weak columns during fine-tuning. This approach allows us to enjoy the benefits of quantization while implementing PEFT.

However, QEFT offers its own unique innovations. First, OWQ suffers from the irregular mixed precision of columns in the weights of linear layers, resulting in low hardware compatibility. In contrast, QEFT achieves a structured mixed precision phenotype based on the novel Offline Global Reordering (OGR), improving hardware compatibility and resulting in substantial speed improvements in both training and inference, as shown in Fig. 1. Additionally, QEFT provides a theoretical framework for selecting weak columns to minimize loss values after fine-tuning. Lastly, despite being implemented differently from LoRA, we validate that QEFT can replace and be applied to applications that previously used LoRA, demonstrating its flexibility. Through various experiments, we showed that QEFT is the state-of-the-art method in terms of inference speed, training speed, and model quality. While it consumes slightly more memory than OWQ, QEFT outperforms it in every other aspect and surpasses other baselines in all areas.

2 Related Work

2.1 Weight-only Quantization of LLMs

Weight-only quantization stands out as one of the most successful optimization methods for LLMs, significantly reducing the model's footprint and mitigating memory bottlenecks during generation, thereby notably accelerating inference. OPTQ (Frantar et al., 2022) pioneered this approach by demonstrating that the OPT-175B model can be quantized to sub-4-bit without notable accuracy degradation. Moreover, this low-precision approach addresses the memory bottleneck, achieving performance benefits on real GPU devices. AWQ (Lin et al., 2023) and TEQ (Cheng et al., 2023) make advances that improve the model quality via fine-grained group-wise quantization.

2.2 Parameter-efficient Fine-tuning (PEFT)

PEFT is designed to minimize fine-tuning costs for LLMs, unlocking their potential to address new problems affordably. LoRA (Hu et al., 2022) is a representative study that freezes pre-trained weights and adds low-rank parameters, which are updated exclusively during fine-tuning. LoRA demonstrates PEFT's potential by showcasing LLMs' remarkable adaptation to unseen tasks with a constrained update scheme.

2.3 Quantization-aware PEFT

QLoRA (Dettmers et al., 2023) expands upon the LoRA concept by incorporating weight quantization, compressing pretrained weight through quantization. While this makes the fine-tuning process more lightweight, the additional path slows down inference performance, as the additional FP16 path cannot be freely merged to quantized base weights.

QA-LoRA (Xu et al., 2023) offers an alternative approach where the updated weight is merged on top of the zero-point of the low-precision weight. This eliminates the need for a high-precision path after fine-tuning unlike QLoRA, and demonstrates comparable fine-tuning quality; however, it still has a large fine-tuning overhead, and the flexibility for the advanced LoRA applications such as PEFT merging is not explored.

2.4 Outlier-aware Weight Quantization

Recently, OWQ (Lee et al., 2024) introduced intra-layer mixed precision quantization scheme for weight only quantization. In OWQ, the goal is to reduce the layer-wise error, which is broken down into the error for the i-th output channel as follows:

$$E_i = ||W_{i,:}X - \hat{W}_{i,:}X||_2^2 \approx \Delta W_{i,:} H \Delta W_{i,:}^T. \quad (1)$$

The Hessian of the weight matrix plays a crucial role in estimating the sensitivity of specific weights. However, the weights in the same output channel share an identical Hessian value, calculated as:

$$H^{(i)} = H = \frac{\partial^2 E_i}{\partial W_{i,:}^2} = 2XX^T. \quad (2)$$

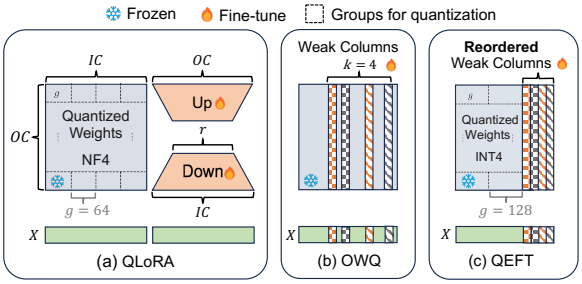


Figure 2: The overview of adaptable quantization includes: (a) QLoRA, (b) OWQ and (c) the proposed QEFT with OGR. $k = 4$ case as an example.

From this formulation, it was observed that activation outliers can significantly increase the sensitivity of specific weight columns even when only weight quantization is applied. In OWQ, they calculated the sensitivity of j-th column due to weight quantization as follows:

$$sensitivity_j = \lambda_j ||\Delta W_{:,j}||_2^2, \tag{3}$$

where λ is a diagonal element of the Hessian. Afterward, they preserved the top-k most sensitive columns (weak columns) in FP16 and compressed only the remaining robust weights into 4 or 3-bit.

OWQ enhances the model quality significantly while only adding an extra 0.01 bits on average. However, its mixed-precision format poses challenges in deployment in both GPU and non-GPU environments, limiting thier practical benefits.

2.5 Weak Column Tuning

OWQ also introduced the concept that enables PEFT with mixed-precision. This idea, known as Weak Column Tuning (WCT), updates the FP16 weak columns in a task-specific manner while freezing the remaining quantized data. WCT in OWQ sustains the benefits of low precision for both inference and fine-tuning, but it also has several limitations; WCT lacks theoretical support to guarantee its optimality in selecting tuning parameters based on weak columns and has only demonstrated feasibility for specific tasks. Furthermore, the versatility of WCT has not yet been validated, making it less favored than LoRA-based approaches. Most importantly, the irregular mixed-precision weights in OWQ poses challenges for acceleration.

3 Detailed Overview of QEFT

In this study, we introduce QEFT, a mixed-precision quantization that achieves higher speed for both inference and fine-tuning and better fine-tuned quality thanks to its hardware-friendly and expressiveness. We begin by detailing QEFT in this section and explain fine-tuning capabilities in Sec. 4. Then, we discuss PEFT merging, an advanced example validating its versatility, in Sec. 5.

3.1 Data Structure and Quantization Process

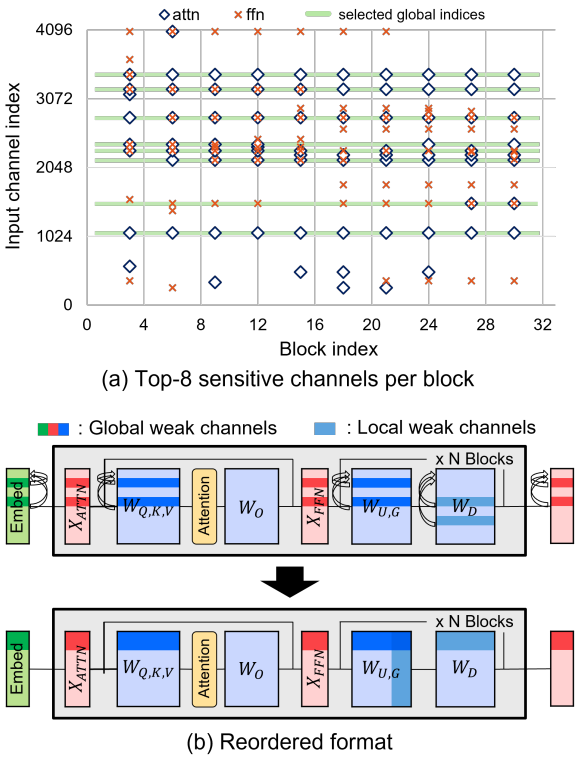
QEFT applies mixed-precision quantization to the dense weights of linear layers in LLMs. After quantization, three data components are generated: the dense low-precision matrix, group-wise quantization parameters, and high-precision weak columns, as depicted in Fig. 2(c). Similar to OWQ, we begin by identifying the k sensitive columns and preserving them in FP16. However, the key difference in implementation is that QEFT employs novel Offline Global Reordering (OGR), as described in Sec. 3.2, ensuring a structured format unlike to the irregularity of OWQ (Fig. 2(b)).

Subsequently, the remaining weights are stored in 4-bit or less. We introduce group-wise quantization (Lin et al., 2023) to further minimize quantization errors from per-channel quantization based OWQ. Therefore, every adjacent g weights share the same quantization parameters, such as scaling factor and zero-point. We perform a grid search for each group to find the quantization parameters that minimize the squared error of weights after truncation. Then, we apply OPTQ (Frantar et al., 2022) using the searched parameters to find the best low-precision mapping. While OPTQ originally relies on channel-wise min-max quantization, ours leverage the benefits of group-wise quantization and truncation (Esser et al., 2019; Li et al., 2021; Nahshan et al., 2021; Wei et al., 2022), resulting in high-quality quantized weights.

3.2 Offline Global Reordering

In OWQ, mixed-precision makes it difficult to accelerate. While the indices of weak columns are predetermined offline, correlated with the location of activation outliers (Fig. 2(b)), the irregular mixed-precision format introduces multiple branches in the decompression process, causing complex implementation and slowdown. Moreover, these characteristics are difficult to support on emerging hardware, such as in-DRAM accelerators (Lee et al., 2021) or NPUs (Intel, 2024), which only support dense computation in general.

To address this limitation, we must revise the data representation for predictability and conti-



(a) Top-8 sensitive channels per block

(b) Reordered format

Figure 3: (a) Weak column indices at all transformer blocks in the Llama-2 7B model, where "attn" indicates input activation of the attention block and "ffn" indicates activation of the feed-forward block. (b) An overview of the offline global reordering.

guity. Previous efforts, such as reordering activations in the normalization layer during inference (Yuan et al., 2023), grouped high-precision weights on one side, separating them from low-precision weights. Although this accelerates computation via dense matrix multiplication, the online reordering incurs additional inference latency, amortizing the benefits of quantization.

To eliminate irregularities without incurring online costs, we introduce a novel concept called Offline Global Reordering (OGR). This idea is motivated by our key observation that outlier channel indices significantly overlap across layers, as shown in Fig. 3(a), which visualizes the layer-wise indices of the top-8 weak columns for each transformer block in the Llama-2 7B model. This overlap occurs because outlier activations propagate through subsequent layers via residual connections, causing weak column indices to align across layers. Based on this observation, we identify and use the common (global) weak columns across the entire network, as depicted in Fig. 3(a). Since weight perturbation can vary significantly across layers, we use only λ_j, instead of $\lambda_j ||\Delta W_{:,j}||_2^2$ in Eq. (3). The

Reorder technique	Group-wise quantization	6task Avg. ↑	4task Avg. ↑	Inference speed (tokens/s) ↑
		51.24	55.46	112
Online		51.24	55.46	127
OGR		51.11	55.19	148
	✓	51.42	55.81	111
OGR	✓	51.55	55.70	146

Table 1: Ablation results of QEFT ($k = 128$) for the few-shot average scores and inference speed on Llama-2 7B. The bottom row represents QEFT. For more details, please see Sec. 6.1.

optimality of this metric is discussed in Sec. 4, and the detailed selection algorithm is in Appendix B.

Once the global weak columns are selected, we can rearrange the weights of the embedding and head layers, as well as the layers within the transformer block, offline, as shown in Fig. 3(b). By globally reordering the model, the weak columns of each linear layer form a structured dense matrix, and their corresponding activations are located contiguously. One exception is the attention output projection layer, or W_O in Fig. 3(b). To maintain the multi-head attention mechanism, reordering is not applicable to the W_O weight. In this case, the mixed-precision format is used without reordering.

Tab. 1 shows the effect of the component proposed in QEFT for Llama-2 7B. For reference, we begin with the OWQ configuration and measure the impact of online reordering and OGR, respectively. The table indicates that online reordering offers limited benefits due to its overhead, whereas OGR significantly accelerates inference. Although most weak columns overlap, utilizing global weak columns may result in subtle accuracy degradation due to the small number of non-overlapping column indices. To realize the Pareto-front solution, we seek to address this degradation and consider using group quantization instead of OWQ's channel-wise quantization. A group size of 128 was used by default and it improves the fine-tuned performance by reducing quantization error, with negligible hardware overhead thanks to our optimized kernel. Due to the increased number of group-wise parameters, a slight increase in memory usage occurs from 3923MB to 4107MB. Results show that using OGR with group quantization is the best option and globally reordered model exhibits nearly the same few-shot scores as the optimal selection at each layer while greatly enhancing inference speed.

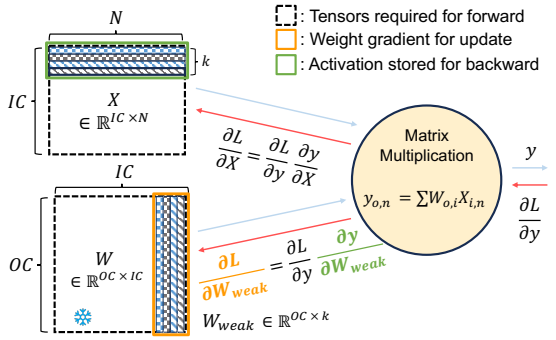


Figure 4: Visualization of the reduction in computation during forward and backward of QEFT in linear layers.

3.2.1 GPU Acceleration Kernel for QEFT

To realize the full potential of QEFT, we developed a customized matrix-vector multiplication GPU kernel tailored for the reordered format. This kernel first processes the quantized dense matrix by dequantizing the weights into FP16 format and multiplying them with the activations. Subsequently, it performs the multiplication of the high-precision dense weights and activations. Thanks to OGR, we can seamlessly apply two dense computations, which are fused into a single kernel in practice. The impact of our customized kernel on inference performance is validated in Sec. 6.4.

3.3 Efficient Backward Computation

Utilizing QEFT provides another significant advantage for fine-tuning. As depicted in Fig. 4, the backward computation of the linear layer involves two GeMM operations: calculating gradients for the input X and the weights W. Unlike LoRA-based approaches, QEFT reduces the GeMM cost by computing gradients only for the rectangular-shaped trainable weights, thus decreasing the overall FLOPs of weight gradients to k/IC. This reduction offers substantial performance benefits during fine-tuning. Additionally, as shown in Fig. 4, we only need to store the subset of activations corresponding to the weak columns. The gradient of weights for weak columns can be computed without requiring the entire activation tensor, reducing the memory footprint to k/IC. A crucial aspect of QEFT is that it uses a structural data representation based on OGR, allowing for easy backward implementation in existing frameworks such as PyTorch (Paszke et al., 2019).

4 Optimal Weak Column Selection

While QEFT is efficient for both fine-tuning and inference, it also offers superior fine-tuning quality. In this section, we provide theoretical support by proving that selecting weak columns as mask for tunable parameters is an optimal strategy for minimizing the loss value after sparse PEFT, under the following conditions: 1. *A fixed budget is allocated to each linear layer.* 2. *The selection is applied at a per-channel granularity.*

Firstly, we formulate the sparse PEFT as follows:

$$\min_{\Delta\theta, M} L(\theta^0 + M\Delta\theta), \tag{4}$$

where $\theta^0 \in \mathbb{R}^{OC\times IC}$ represents the pre-trained weights, and OC and IC represent the output and input channel dimensions, respectively. $\Delta\theta \in \mathbb{R}^{OC\times IC}$ represents the updated weights, L represents the target loss function, and $M \in \mathbb{R}^{IC\times IC}$ represents the channel-wise parameter mask, where $M_{i,j} = 0$ if $i \neq j$ or $M_{i,i} \in \{0, 1\}$ otherwise. To maximize the effect of fine-tuning, we need to select an appropriate M that can minimize the loss. According to the second-order approximation method of (Fu et al., 2023), we can find out the optimal mask based on the magnitude of the gradient.

Theorem.

$$\textit{if } \hat{M}_{ii} = \mathbb{1}\left(\sum_{j=1}^{m} \mathbb{1}\left(\left|\frac{\nabla L(\theta^0)_i^2}{h_i}\right| > \left|\frac{\nabla L(\theta^0)_j^2}{h_j}\right|\right) \geq m-k\right),$$

where $\nabla L(\theta^0)_i$ *is the* i*-th element of* $\nabla L(\theta^0)$*, then*

$$\inf_{\Delta\theta} L(\theta^0 + \hat{M}\Delta\theta) \leq \inf_{\Delta\theta, \|M\|_0 = k;} L(\theta^0 + M\Delta\theta).$$

This theorem states that the mask $\hat{M}$ minimizing the infimum of loss can be constructed by selecting the top k indices in order of largest $|\nabla L(\theta^0)_i^2|$ values. In the constraints of the channel-wise parameter mask, by selecting the indices with the largest $|\nabla L(\theta^0)_{:,i}^2|$ values, we can construct $\hat{M}$ that minimizes the loss after fine-tuning among candidates.

In QEFT, tunable weak columns are selected by Eq. (3), which is based on λ_i and the weight perturbation. Meanwhile, the gradient of the weight in the linear layer is calculated by the chain rule:

$$\nabla L(\theta) = \frac{\partial L}{\partial \theta} = \frac{\partial L}{\partial y}\frac{\partial y}{\partial \theta} = \frac{\partial L}{\partial y}X^T, \tag{5}$$

where X represents the activation and $y = \theta X$. Most importantly, the presence of activation outliers causes both weak column selection (Eq. (3))

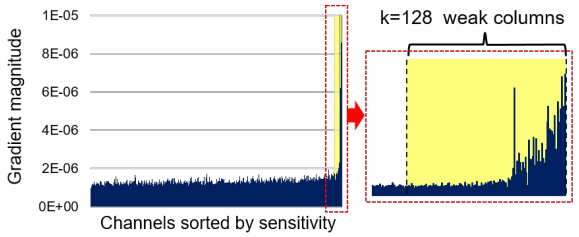


Figure 5: Channel-wise sensitivity and magnitude of the gradient in Llama-2 7B model. The yellow box indicates the selected weak columns for $k = 128$ case.

and weight gradient to be dominated by the activation, thus the selection metric of QEFT is also valid for selecting columns with the largest $|\nabla L(\theta^0)^2_{:,i}|$.

Fig. 5 illustrates the correlation between Eq. (3) and $|\nabla L(\theta^0)^2_{:,i}|$. Sorting the channels using quantization sensitivity reveals that the top-k channels (weak columns) also represent the columns with the largest gradient magnitude. This implies that although we select weak columns considering the quantization sensitivity, fine-tuning quality is also accounted for.

5 Advanced Application: PEFT Merging

QEFT is designed for efficient inference and fine-tuning, but it also needs to be general enough to be an alternative to the LoRA-based approach. To validate this, we applied QEFT to an advanced application of LoRA known as PEFT merging (Lee et al., 2023). Fig. 6 provides an overview of this process. As shown in the figure, the LoRA adapter is fine-tuned using the Open-Platypus dataset (Lee et al., 2023) on the Llama-2 model. After fine-tuning, the updated weights are transferred to the StableBeluga (Mahan et al., 2023) model, which was also initialized with Llama-2 but fine-tuned on a different dataset, resulting in the creation of Stable-Platypus2. Despite potential differences in semantics, Stable-Platypus2 surprisingly exhibits better quality than each model before the merge.

To assess the merging ability of QEFT, we also attempt a similar approach, called QEFT merging. We generate the quantized Llama-2 model and fine-tune the weak columns for the Open-Platypus dataset. The updates of the weak columns ($\Delta = B - A$) are then merged into the StableBeluga model. If the target model is a full-precision model, we add the update according to the weak column index. For the quantized StableBeluga, we add the update to the corresponding weak columns. We show that QEFT merging works surprisingly

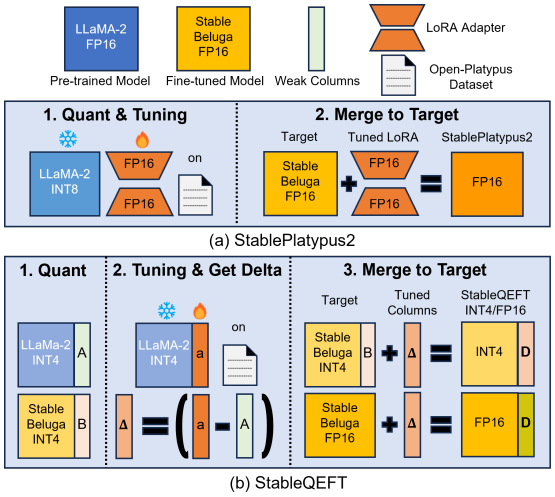


Figure 6: An overview of LoRA-based PEFT merging and its QEFT counterparts, QEFT merging. (a) Target model + LoRA case. (b) Target model + QEFT case.

well in Sec. 6.3, validating its generality.

6 Experiments

6.1 Experiments Setting

To demonstrate the superiority of QEFT, we conducted extensive analysis. The fine-tuning environment adheres to the setup of a baseline, Platypus (Lee et al., 2023). We utilized the Open-Platypus dataset for fine-tuning, specially filtered to remove duplicates and redundancy among 11 open-source datasets. Given that the Open-Platypus dataset focuses on STEM and logic question domains, we also adopted their evaluation method, which includes few-shot tasks from the open-llm-leaderboard (Beeching et al., 2023).

Following the recent version of the leaderboard, we report scores for 6 tasks (MMLU (Hendrycks et al., 2020), HellaSwag (Zellers et al., 2019), ARC-c (Clark et al., 2018), TruthfulQA (Lin et al., 2022), Winogrande (Sakaguchi et al., 2021), and GSM8k (Cobbe et al., 2021)), along with their average, to assess fine-tuning performance. Additionally, we report the average score of four tasks (MMLU, HellaSwag, ARC-c, and TruthfulQA) used for evaluation in previous studies (Lee et al., 2023; Xu et al., 2023). We utilized lm-eval-harness (Gao et al., 2023) to measure few-shot accuracy.

We used Adamw optimizer with batch size of 16. We used constant learning rate of 1e-5 and 5e-6 for $k = 16$ and 128, respectively. We observed the overfitting issue of OPTQ reconstruction in the 70B model, which is also found in the previous

Model	Bits	Base Size	Tunable Params.	Tuning Time ↓	Inference Speed ↑ (tokens/s)	speedup	MMLU	Hella Swag	ARC-c	Truthful QA	Wino grande	GSM8k	6task Avg. ↑	4task Avg. ↑
Llama-2 **7B**	16	12.9GB	-	-	93	1×	46.54	78.63	52.99	38.96	73.64	14.63	50.90	54.28
LoRA	16	12.9GB	160M	1.9h	93	1×	48.18	78.32	55.29	41.78	74.27	1.67	49.92	55.89
Platypus	8	6.7GB	23M	7.0h	11	0.12×	49.93	78.74	54.44	42.70	74.79	2.84	50.57	**56.45**
QLoRA	4	3.7GB	160M	3.5h	17	0.18×	48.44	77.75	54.44	41.71	74.12	1.48	49.65	55.58
QA-LoRA	4	4.4GB	89M	5.2h	47	0.51×	48.61	78.37	53.11	41.28	73.68	14.14	51.50	55.41
OWQ (k=16)	4	3.6GB	22M	1.7h	119	1.28×	46.59	78.00	52.26	40.86	73.16	12.58	50.58	54.43
OWQ (k=128)	4	3.9GB	174M	1.7h	112	1.20×	48.24	78.01	54.14	41.47	73.32	12.25	51.24	55.46
QEFT (k=16)	4	3.8GB	22M	**1.7h**	**148**	**1.59**×	48.70	78.21	53.54	41.96	73.29	12.17	51.31	55.60
QEFT (k=128)	4	4.1GB	174M	**1.7h**	**146**	**1.57**×	49.02	78.21	53.80	41.77	73.56	12.93	**51.55**	55.70
Llama-2 **13B**	16	24.8GB	-	-	52	1×	55.42	82.19	59.64	36.90	76.09	21.38	55.27	58.54
LoRA	16	24.8GB	250M	2.7h	52	1×	55.59	82.24	60.92	45.64	76.44	6.79	54.60	**61.10**
Platypus	8	12.7GB	36M	10.7h	8	0.15×	56.70	82.32	60.37	42.16	75.85	10.66	54.67	60.38
QLoRA	4	6.9GB	250M	6.1h	13	0.25×	55.86	81.76	59.56	44.30	76.44	8.57	54.41	60.37
QA-LoRA	4	8.2GB	140M	8.9h	39	0.75×	56.66	81.95	61.22	41.75	76.91	22.51	56.84	60.39
OWQ (k=16)	4	6.7GB	34M	2.6h	80	1.54×	56.03	81.81	60.32	40.81	76.00	20.32	55.88	59.74
OWQ (k=128)	4	7.2GB	273M	2.7h	76	1.46×	57.30	82.05	60.20	42.24	76.96	22.18	56.82	60.45
QEFT (k=16)	4	7.1GB	34M	**2.5h**	**101**	**1.94**×	56.40	81.71	61.86	42.99	76.24	23.13	57.05	60.74
QEFT (k=128)	4	7.6GB	273M	**2.6h**	**98**	**1.88**×	56.80	82.01	62.33	42.46	77.51	22.56	**57.28**	60.90
Llama-2 **70B**	16	131.6GB	-	-	11*	1×	69.83	87.33	67.32	44.92	83.74	54.06	67.87	67.35
Platypus †	8	66.3GB	141M	88h	4	0.36×	70.04	87.02	70.14	51.13	83.74	54.89	69.49	69.58
QLoRA	4	34.7GB	828M	28.1h	7	0.64×	69.82	87.06	69.03	51.05	84.93	55.04	69.49	69.24
QA-LoRA	4	41.8GB	442M	41.2h	20	1.82 ×	70.20	87.32	69.50	47.69	83.94	53.18	68.51	68.54
OWQ (k=16)	4	33.9GB	107M	11.0h	23	2.09×	69.97	86.91	69.28	51.56	84.17	54.28	69.36	69.43
OWQ (k=128)	4	35.3GB	860M	11.2h	22	2.00×	70.25	86.89	70.31	50.02	84.53	53.15	69.19	69.37
QEFT (k=16)	4	35.8GB	107M	**10.9h**	**29**	**2.64**×	70.49	86.85	70.05	50.52	83.90	56.03	**69.64**	69.48
QEFT (k=128)	4	37.3GB	860M	**11.1h**	**28**	**2.55**×	70.51	86.88	69.97	51.15	83.98	54.44	69.49	**69.63**

Table 2: Comparison of PEFT methods for various few-shot tasks. The model group is divided into 7B, 13B, and 70B by horizontal double lines. Among the average scores, we **bold** the best score and underline the second and third-best scores. † denotes that the accuracy was measured using an official checkpoint. The training cost is measured by A100 GPU hours. * denotes using 2 GPUs, as FP16 70B model causes OOM on single GPU.

work (Wu et al., 2023). Therefore, we utilized simple round-to-nearest quantization instead of OPTQ for 4-bit Llama-2 70B results. Please refer to Appendix A for the detailed experiment setups.

6.2 Overall Fine-tuning Results

In order to compare the superiority of QEFT, we selecte five representative counterparts: LoRA, Platypus, QLoRA, QA-LoRA, and OWQ. Platypus (Lee et al., 2023) utilizes 8-bit quantization for the base model and integrates the LoRA module only into the FFNs. QLoRA/QA-LoRA and OWQ are included for quality and performance comparisons using 4-bit quantization. QLoRA/QA-LoRA applies the LoRA module to all linear layers, while OWQ retains k weak columns as FP16 for all linear layers. It's important to note that $k/2 \simeq r$ in terms of tunable parameters, as each LoRA module employs two $d \times r$ adapters.

We reproduced all results from our control groups (LoRA, Platypus, QLoRA, QA-LoRA, and OWQ), excluding Platypus 70B due to its resource-intensive nature (requiring 8xA100 GPUs) and lengthy training time. Instead, we utilized the officially provided pre-trained weights of Platypus 70B from Huggingface. Since OWQ does not provide tuning code, we implemented it based on our setup. Therefore, the tuning of OWQ was also accelerated by using the QEFT's customized code. In addition, different from inference, computations are mostly compute-bound matrix multiplication during fine-tuning. In this case, the speed gain of OGR just comes from the simple dequantization process, resulting in negligible benefit ($\sim$ 0.1h reduction of training time) compared to OWQ.

Experimental results are detailed in Tab. 2. QEFT clearly outperforms the other quantization-aware PEFTs in the 13B model. However, Platypus demonstrates the best tuning performance in the 7B model, primarily because accuracy degradation due to quantization is dominant for the smaller model. Yet, when considering the base size (6.7GB vs. 4.1GB), QEFT's performance stands out. Moreover, even in the 7B case, QEFT with $k = 128$ shows better results than other 4-bit baselines, and in the 13B case, QEFT with just a budget of $k = 16$ exhibits tuning performance outperforms others. Interestingly, Platypus, LoRA, and QLoRA consistently display low GSM8k scores on 7B and 13B, indicating they might be overly tuned for conventional 4 benchmarks.

Platypus reports that their 70B model fine-tuning

Model	Type	Bits	MMLU	HellaSwag	ARC-c	TruthfulQA	4tasks Avg. ↑
Llama-2 13B	PT	16	55.42	82.19	59.64	36.90	58.54
StableBeluga-13B	FT	16	57.65	82.35	61.95	49.21	62.79
Stable-Platypus2-13B	target + LoRA	16	58.15	82.31	**62.54**	52.38	63.85
Stable-QEFT-13B	target + QEFT	4	58.32	81.86	62.20	52.25	63.66
Stable-QEFT-13B		16	**58.97**	**82.53**	62.46	**53.71**	**64.42**
OpenOrcaxOpenChat-Preview2	FT	16	58.52	83.09	62.63	50.57	63.70
OpenOrca-Platypus2-13B	target + LoRA	16	59.46	**83.21**	62.88	52.69	64.56
OpenOrca-QEFT-13B	target + QEFT	4	59.18	82.51	63.48	53.40	64.64
OpenOrca-QEFT-13B		16	**59.47**	83.10	**63.74**	**54.31**	**65.16**

Table 3: PT, FT, and target + X denote pre-trained model, full fine-tuning, and parameter-efficient fine-tuning using X, respectively. Among the PEFT merging results, we **bold** the best score and underline the second-best score.

required 4 × A100 GPUs for 22 hours. In contrast, QEFT can be completed in 11 hours with just a single A100 GPU. This represents approximately an 8-fold acceleration in terms of GPU hours, highlighting the memory-time efficiency feature of the proposed method. Despite having only 1/8 of the training cost, the fine-tuned quality of QEFT-based model is comparable to or better than the Platypus-70B, significantly surpassing the baseline Llama-2 70B. Compared to other 4-bit methods, QEFT consistently shows better accuracy. In particular, QEFT has a much lower tuning time than LoRA-based approaches.

6.3 PEFT Merging Results

Tab. 3 presents the PEFT merging results for two fully fine-tuned models, StableBeluga and OpenOrca (Lian et al., 2023). Following the objective of merging strategy from the Platypus study, we report the scores of the four tasks (MMLU, HellaSwag, ARC-c, and TruthfulQA) to assess complementary effects. Although QEFT merging only modifies the weight columns corresponding to weak columns of the fully fine-tuned models, all QEFT merging cases increases MMLU, ARC, and TruthfulQA scores together; thus, merging tuned weak columns enhances both reasoning ability and knowledge capability. Furthermore, QEFT merging with a quantized target model (indicated as 4-bits in the table) demonstrates comparable accuracy to FP16 LoRA merging. In this scenario, we benefit from a reduced merged model size and faster inference from the QEFT format. Both merging techniques prove highly beneficial for enhancing model quality. This observation confirms that QEFT aligns well with the idea of PEFT merging.

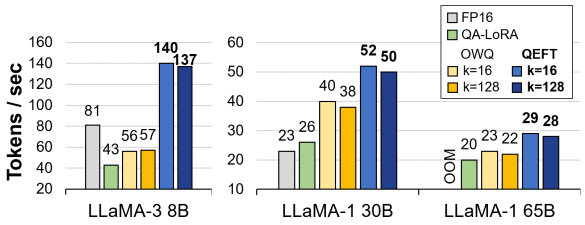


Figure 7: The number of generated tokens per second in auto-regressive generation scenario at the single batch.

6.4 Inference Acceleration

In this section, we demonstrate performance benefits of QEFT in auto-regressive generation scenarios. We benchmark inference speed on the LLaMA families on A100 80GB GPU, using the full-precision model and different tuning methods as a baseline. In this experiment, we utilize the optimized multi-head attention kernel and layer normalization kernel of FasterTransformer (NVIDIA, 2023) for all methods except QA-LoRA, as they use AutoGPTQ (AutoGPTQ, 2024) framework. Please refer to Appendix A.2 for details. We report the results as the median of generated tokens per second, generating 256 tokens at batch size 1.

As shown in the Tab. 2, the low-rank path in LoRA-based methods introduces a bottleneck in inference, notably reducing overall speed. Additionally, as described in Tab. 2 and Fig. 7, even though OWQ also utilizes their optimized kernel, QEFT consistently achieved speedups of about 30% across from OWQ all model sizes in Llama-1/3. This improvement stems from eliminating the irregular memory access, a major contributor to the overhead of mixed-precision operations, through OGR. In particular, QEFT is about 2.4x faster on the Llama-3 8B model compared to OWQ, because QEFT's kernel exhibits high parallel throughput even for the model with small linear layers and group query attention.

7 Conclusion

The storage and computation demands of LLMs present significant barriers to widespread adoption. While quantization and PEFT optimize inference and fine-tuning, respectively, their harmonization is overlooked. In this work, we introduce QEFT, designed to ensure fine-tuning compatibility while prioritizing hardware compatibility. Our experiments confirm that QEFT achieves the fastest fine-tuning and yields the highest accuracy. Moreover, QEFT shows superior performance in inference, highlighting its excellence across all aspects.

Limitations

Although model quantization and parameter-efficient fine-tuning can serve as regularizers during training (AskariHemmat et al., 2022; Fu et al., 2023), an explicit regularization mechanism is often beneficial for stabilizing fine-tuning outcomes. In practice, additional regularizers like Dropout (Srivastava et al., 2014) are employed in LoRA modules to ensure stable training. However, in QEFT, the Dropout-like regularizer is not currently incorporated. While our experiments did not exhibit evident signs of overfitting, this limitation may lead to unaddressed overfitting concerns depending on the specific task. As a future direction, we aim to explore potential candidates for regularization.

In terms of performance, QEFT requires time to convert FP16 to low-precision representation. Typically, the conversion process for the Llama-2 13B model takes about an hour and further details are in the Tab. 8. We did not incorporate this conversion cost into Tab. 2 because it is a one-time expense, and its cost can be amortized by reusing the quantized model for multiple downstream tasks. However, if LLM capacities are further increased, it may be necessary to consider this conversion cost.

While we can determine the global weak columns and enjoy the benefit of OGR, the 'out_proj' weight cannot be reordered either offline or via equivalent out channel reordering of previous layers. Therefore, we have to specially handle the reordering of 'out_proj' on the fly, utilizing the customized kernel based on OWQ. This hinders the advantage of QEFT's high-throughput generation system.

Acknowledgement

This work was supported by Institute of Information & communications Technology Planning & Evaluation (IITP) grant funded by the Korea government (MSIT) (RS-2023-00213611, RS-2024-00415602, RS-2024-00396013).

References

MohammadHossein AskariHemmat, Reyhane Askari Hemmat, Alex Hoffman, Ivan Lazarevich, Ehsan Saboori, Olivier Mastropietro, Sudhakar Sah, Yvon Savaria, and Jean-Pierre David. 2022. Qreg: On regularization effects of quantization. *arXiv preprint arXiv:2206.12372*.

AutoGPTQ. 2024. Autogptq. https://github.com/AutoGPTQ/AutoGPTQ.

Edward Beeching, Clémentine Fourrier, Nathan Habib, Sheon Han, Nathan Lambert, Nazneen Rajani, Omar Sanseviero, Lewis Tunstall, and Thomas Wolf. 2023. Open llm leaderboard. https://huggingface.co/spaces/HuggingFaceH4/open_llm_leaderboard.

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

Wenhua Cheng, Yiyang Cai, Kaokao Lv, and Haihao Shen. 2023. Teq: Trainable equivalent transformation for quantization of llms. *arXiv preprint arXiv:2310.10944*.

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. *arXiv preprint arXiv:1803.05457*.

Karl Cobbe, Vineet Kosaraju, Mohammad Bavarian, Mark Chen, Heewoo Jun, Lukasz Kaiser, Matthias Plappert, Jerry Tworek, Jacob Hilton, Reiichiro Nakano, et al. 2021. Training verifiers to solve math word problems. *arXiv preprint arXiv:2110.14168*.

Tim Dettmers, Artidoro Pagnoni, Ari Holtzman, and Luke Zettlemoyer. 2023. Qlora: Efficient finetuning of quantized llms. In *Thirty-seventh Conference on Neural Information Processing Systems*.

Steven K Esser, Jeffrey L McKinstry, Deepika Bablani, Rathinakumar Appuswamy, and Dharmendra S Modha. 2019. Learned step size quantization. In *International Conference on Learning Representations*.

Elias Frantar and Dan Alistarh. 2023. Sparsegpt: Massive language models can be accurately pruned in one-shot. In *International Conference on Machine Learning*, pages 10323–10337. PMLR.

Elias Frantar, Saleh Ashkboos, Torsten Hoefler, and Dan Alistarh. 2022. Optq: Accurate quantization for generative pre-trained transformers. In *The Eleventh International Conference on Learning Representations*.

Zihao Fu, Haoran Yang, Anthony Man-Cho So, Wai Lam, Lidong Bing, and Nigel Collier. 2023. On the effectiveness of parameter-efficient fine-tuning. *Proceedings of the AAAI Conference on Artificial Intelligence*, 37(11):12799–12807.

Leo Gao, Jonathan Tow, Baber Abbasi, Stella Biderman, Sid Black, Anthony DiPofi, Charles Foster, Laurence Golding, Jeffrey Hsu, Alain Le Noac'h, Haonan Li, Kyle McDonell, Niklas Muennighoff, Chris Ociepa, Jason Phang, Laria Reynolds, Hailey Schoelkopf, Aviya Skowron, Lintang Sutawika, Eric Tang, Anish Thite, Ben Wang, Kevin Wang, and Andy Zou. 2023. A framework for few-shot language model evaluation.

Shibo Hao, Tianyang Liu, Zhen Wang, and Zhiting Hu. 2024. Toolkengpt: Augmenting frozen language models with massive tools via tool embeddings. *Advances in neural information processing systems*, 36.

Dan Hendrycks, Collin Burns, Steven Basart, Andy Zou, Mantas Mazeika, Dawn Song, and Jacob Steinhardt. 2020. Measuring massive multitask language understanding. In *International Conference on Learning Representations*.

Coleman Hooper, Sehoon Kim, Hiva Mohammadzadeh, Michael W Mahoney, Yakun Sophia Shao, Kurt Keutzer, and Amir Gholami. 2024. Kvquant: Towards 10 million context length llm inference with kv cache quantization. *arXiv preprint arXiv:2401.18079*.

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 *International Conference on Machine Learning*, 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 *ICLR*. OpenReview.net.

Intel. 2024. Intel gaudi 3 ai accelerator white paper. https://www.intel.com/content/www/us/en/content-details/817486/intel-gaudi-3-ai-accelerator-white-paper.html. Accessed: 2024-04-15.

Sanghyeon Kim, Hyunmo Yang, Younghyun Kim, Youngjoon Hong, and Eunbyung Park. 2023. Hydra: Multi-head low-rank adaptation for parameter efficient fine-tuning. *CoRR*, abs/2309.06922.

Ariel Lee, Cole Hunter, and Nataniel Ruiz. 2023. Platypus: Quick, cheap, and powerful refinement of llms. In *NeurIPS 2023 Workshop on Instruction Tuning and Instruction Following*.

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

Sukhan Lee, Shin-haeng Kang, Jaehoon Lee, Hyeonsu Kim, Eojin Lee, Seungwoo Seo, Hosang Yoon, Seungwon Lee, Kyounghwan Lim, Hyunsung Shin, et al. 2021. Hardware architecture and software stack for pim based on commercial dram technology: Industrial product. In *2021 ACM/IEEE 48th Annual International Symposium on Computer Architecture (ISCA)*, pages 43–56. IEEE.

Yaniv Leviathan, Matan Kalman, and Yossi Matias. 2023. Fast inference from transformers via speculative decoding. In *International Conference on Machine Learning*, pages 19274–19286. PMLR.

Yuhang Li, Ruihao Gong, Xu Tan, Yang Yang, Peng Hu, Qi Zhang, Fengwei Yu, Wei Wang, and Shi Gu. 2021. Brecq: Pushing the limit of post-training quantization by block reconstruction. In *ICLR*.

Wing Lian, Bleys Goodson, Eugene Pentland, Austin Cook, Chanvichet Vong, and "Teknium". 2023. Openorca: An open dataset of gpt augmented flan reasoning traces. https://https://huggingface.co/Open-Orca/OpenOrca.

Ji Lin, Jiaming Tang, Haotian Tang, Shang Yang, Xingyu Dang, and Song Han. 2023. Awq: Activation-aware weight quantization for llm compression and acceleration. *arXiv preprint arXiv:2306.00978*.

Stephanie Lin, Jacob Hilton, and Owain Evans. 2022. Truthfulqa: Measuring how models mimic human falsehoods. In *Proceedings of the 60th Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pages 3214–3252.

Haokun Liu, Derek Tam, Mohammed Muqeeth, Jay Mohta, Tenghao Huang, Mohit Bansal, and Colin A Raffel. 2022. Few-shot parameter-efficient fine-tuning is better and cheaper than in-context learning. *Advances in Neural Information Processing Systems*, 35:1950–1965.

Dakota Mahan, Ryan Carlow, Louis Castricato, Nathan Cooper, and Christian Laforte. 2023. Stable beluga models.

Xupeng Miao, Gabriele Oliaro, Zhihao Zhang, Xinhao Cheng, Zeyu Wang, Rae Ying Yee Wong, Zhuoming Chen, Daiyaan Arfeen, Reyna Abhyankar, and Zhihao Jia. 2023. Specinfer: Accelerating generative llm serving with speculative inference and token tree verification. *arXiv preprint arXiv:2305.09781*, 1(2):4.

Yury Nahshan, Brian Chmiel, Chaim Baskin, Evgenii Zheltonozhskii, Ron Banner, Alex M Bronstein, and Avi Mendelson. 2021. Loss aware post-training quantization. *Machine Learning*, 110(11-12):3245–3262.

NVIDIA. 2023. Fastertransformer. https://github.com/NVIDIA/FasterTransformer.

NVIDIA. 2024. Trt-llm. https://github.com/NVIDIA/TensorRT-LLM.

Adam Paszke, Sam Gross, Francisco Massa, Adam Lerer, James Bradbury, Gregory Chanan, Trevor Killeen, Zeming Lin, Natalia Gimelshein, Luca Antiga, et al. 2019. Pytorch: An imperative style, high-performance deep learning library. *Advances in neural information processing systems*, 32.

Yujia Qin, Shihao Liang, Yining Ye, Kunlun Zhu, Lan Yan, Yaxi Lu, Yankai Lin, Xin Cong, Xiangru Tang, Bill Qian, et al. 2023. Toolllm: Facilitating large language models to master 16000+ real-world apis. *arXiv preprint arXiv:2307.16789*.

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.

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

Wenqi Shao, Mengzhao Chen, Zhaoyang Zhang, Peng Xu, Lirui Zhao, Zhiqian Li, Kaipeng Zhang, Peng Gao, Yu Qiao, and Ping Luo. 2023. Omniquant: Omnidirectionally calibrated quantization for large language models. In *The Twelfth International Conference on Learning Representations*.

Nitish Srivastava, Geoffrey Hinton, Alex Krizhevsky, Ilya Sutskever, and Ruslan Salakhutdinov. 2014. Dropout: a simple way to prevent neural networks from overfitting. *The journal of machine learning research*, 15(1):1929–1958.

Mingjie Sun, Zhuang Liu, Anna Bair, and J Zico Kolter. 2023. A simple and effective pruning approach for large language models. In *The Twelfth International Conference on Learning Representations*.

Hugo Touvron, Thibaut Lavril, Gautier Izacard, Xavier Martinet, Marie-Anne Lachaux, Timothée Lacroix, Baptiste Rozière, Naman Goyal, Eric Hambro, Faisal Azhar, et al. 2023. Llama: Open and efficient foundation language models. *arXiv preprint arXiv:2302.13971*.

Yiming Wang, Yu Lin, Xiaodong Zeng, and Guannan Zhang. 2023. Multilora: Democratizing lora for better multi-task learning. *CoRR*, abs/2311.11501.

Xiuying Wei, Ruihao Gong, Yuhang Li, Xianglong Liu, and Fengwei Yu. 2022. Qdrop: Randomly dropping quantization for extremely low-bit post-training quantization. In *ICLR*.

Xiaoxia Wu, Haojun Xia, Stephen Youn, Zhen Zheng, Shiyang Chen, Arash Bakhtiari, Michael Wyatt, Yuxiong He, Olatunji Ruwase, Leon Song, et al. 2023. Zeroquant (4+ 2): Redefining llms quantization with a new fp6-centric strategy for diverse generative tasks. *arXiv preprint arXiv:2312.08583*.

Yuhui Xu, Lingxi Xie, Xiaotao Gu, Xin Chen, Heng Chang, Hengheng Zhang, Zhengsu Chen, XIAOPENG ZHANG, and Qi Tian. 2023. Qa-lora: Quantization-aware low-rank adaptation of large language models. In *The Twelfth International Conference on Learning Representations*.

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

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

Elad Ben Zaken, Yoav Goldberg, and Shauli Ravfogel. 2022. Bitfit: Simple parameter-efficient fine-tuning for transformer-based masked language-models. In *Proceedings of the 60th Annual Meeting of the Association for Computational Linguistics (Volume 2: Short Papers)*, pages 1–9.

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

Susan Zhang, Stephen Roller, Naman Goyal, Mikel Artetxe, Moya Chen, Shuohui Chen, Christopher Dewan, Mona Diab, Xian Li, Xi Victoria Lin, et al. 2022. Opt: Open pre-trained transformer language models. *arXiv preprint arXiv:2205.01068*.

A Experimental Details

A.1 Experimental Details of Fine-tuning

We used lm-eval-harness commit version b281b09 for a fair comparison. GSM8k is the only task involving generation among the tasks, so we used a batch size of 1 for evaluating GSM8k to minimize the padding effect. Other few-shot tasks were evaluated using the maximum available batch size. We followed the evaluation methods and datasets provided by lm-eval-harness (Gao et al., 2023), for example, HellaSwag was evaluated using 10k validation data and GSM8k was evaluated using 1.3k test data.

We utilized gradient checkpointing and gradient accumulation to reduce fine-tuning memory usage. Although these options make training slower, our tuning costs (GPUh) were measured using these options by default. For QEFT tuning of the Llama-2 70B model, we used the max_grad_norm value of 0.0. For the fine-tuning configurations of our control groups, we mostly followed the original training strategy and experimental configurations of each control group. Additionally, The QA-LoRA model is quantized using 128 calibration samples extracted from the C4 dataset. We report the 2 seed average score for 7B/13B in Tab. 2. Please refer to the Tab. 4 for detailed hyperparameters.

A.2 Experimental Details of the Generation Benchmark

QA-LoRA exploits the AutoGPTQ (AutoGPTQ, 2024) library for their fine-tuning and inference. However, we found that the datatype of AutoGPTQ's zero point is integer while QA-LoRA utilizes floating point zero point. This difference could potentially lead to a decrease in the kernel speed of QA-LoRA when considering full functionality. QEFT's matrix vector multiplication kernel implementation is based on TensorRT-LLM (NVIDIA, 2024).

B Algorithm for Global Weak Column Index Selection

In the Sec. 3.2, we analyzed the location of weak columns and proposed efficient reordering by taking advantage of the fact that they overlap a lot. Nevertheless, there are weak columns that do not overlap. To determine optimal global weak column index within limited budget (top-k $<$ union of weak columns), we propose Algorithm 1 based on their sensitivity value.

Algorithm 1 Global index selection

```
# d: hidden state dimension of models.
# k: number of outlier channel
# X: encoded sequences. shape :[b, s, d]

def compute_sensitivity(layer, X):
    H = (2*X @ X.T).mean(dim=0)
    sensitivity = H.diag()
    top_indices = sensitivity.topk(k).indices
    return sensitivity, top_indices

s_global = torch.zeros(d)

for block in blocks:
   for layer in block:
      s_local, ids = compute_sensitivity(layer, X)
      s_global[ids] += s_local[ids] / s_local.mean()

global_indices = torch.topk(s_global, k).indices
```

C Additional Experiments

C.1 Few-shot Results on 3-bit Settings

we added results for 3-bit settings to Tab. 5. The accuracy gap between QEFT and other methods increases, except for a single case: QA-LoRA with Llama-2 13B. Two factors mainly affect fine-tuned accuracy: (1) mask selection and corresponding fine-tuning ability, and (2) quality of the frozen quantized weights.

In the case of OWQ and QEFT, the fine-tuning ability of (1) is the same according to the convergence analysis in Section 3. Therefore, the difference in accuracy is caused by (2), where group-wise quantization of QEFT makes a gap in quantization quality. Thus, it is also natural that the accuracy gap increases in 3-bit, where quantization quality is even more important.

On the other hand, QA-LoRA utilizes a group size of 32, so generally QA-LoRA has a better quantization quality regarding (2) while having a large storage overhead of base size in return, as clearly shown in the table. Nevertheless, as QEFT guarantees lower convergence, QEFT shows better accuracy in most 4-bit and 3-bit cases as QEFT's (1) dominates QA-LoRA's (2). However, as quantization quality becomes more important in 3-bit, we assume a single reversed result occurred for 13B.

In this case, the base size of QA-LoRA is 6.68GB, which is about 12% larger than QEFT's 6.05GB. However, it is difficult to match all other configurations and conditions (e.g. tuning cost, base size, inference speed, etc.).

C.2 Comparison of Quantization Methods

Selecting an adequate quantization method is important as it is directly connected to the post-tuned

Hyper-parameter	QEFT		LoRA	Platypus	QLoRA		QA-LoRA	
	$k = 16$	$k = 128$	7B/13B	7B/13B	7B/13B	70B	7B/13B	70B
group_size	128		-	per-tensor	64		32	
LR	1e-5	5e-6	2e-4	4e-4	2e-4	1e-4	2e-5	1e-5
Dropout	-	-	0.1	0.05	0.1	0.05	0	0
Scheduler	constant	constant	constant	cosine	constant	constant	constant	constant
warmup steps	0	0	0	100	0	0	0	0
Low rank (r)	-	-	64	16	64	64	64	64
LoRA modules	-	-	all	FFN	all	all	all	all
max_grad_norm	0.3	0.3	0.3	1.0	0.3	0.3	0.3	0.3
per_device_train_batch_size	1							
gradient_accumulation_steps	16							

Table 4: The experimental configurations and hyperparameters for fine-tuning.

Model	**Base Size**	**Tunable Params.**	**MMLU**	**Hella Swag**	**ARC-c**	**Truthful QA**	**Wino grande**	**GSM8k**	**6task Avg. ↑**	**4task Avg. ↑**
Llama-2 **7B**										
QA-LoRA	3.59GB	89M	44.97	76.68	49.91	40.75	71.98	11.90	49.36	53.08
OWQ (k=128)	3.15GB	174M	45.06	76.22	52.47	39.31	72.85	10.54	<u>49.41</u>	<u>53.27</u>
QEFT (k=128)	3.33GB	174M	45.07	76.88	52.86	42.86	72.45	10.85	**50.16**	**54.42**
Llama-2 **13B**										
QA-LoRA	6.68GB	140M	55.75	81.16	59.04	42.77	75.77	20.62	**55.85**	**59.68**
OWQ (k=128)	5.69GB	273M	55.63	80.69	57.94	42.10	76.01	18.12	55.08	59.09
QEFT (k=128)	6.05GB	273M	56.18	80.67	58.41	42.54	75.89	20.74	<u>55.74</u>	<u>59.45</u>
Llama-2 **70B**										
QA-LoRA	33.64GB	442M	69.64	86.65	68.09	46.89	83.58	52.08	67.82	67.82
OWQ (k=128)	27.14GB	860M	69.05	86.08	67.92	49.81	83.66	50.72	<u>67.87</u>	<u>68.22</u>
QEFT (k=128)	29.10GB	860M	70.07	85.97	68.17	52.55	83.66	51.18	**68.60**	**69.19**

Table 5: Comparison of PEFT methods for various few-shot tasks on 3-bit base settings. The model group is divided into 7B, 13B, and 70B by horizontal lines. Among the average scores, we **bold** the best score and <u>underline</u> the second scores.

	WikiText-2 ↓		Avg. of 6 tasks ↑	
	7B	13B	7B	13B
RTN	5.25	**4.65**	50.04	56.43
OWQ	**5.22**	4.66	**51.16**	**56.82**

Table 6: Comparison of quantization methods of the base model.

k	**8**	**16**	**32**	**64**	**128**
lr ($\times 10^{-5}$)	1.4	1.0	0.7	0.5	0.4
Avg	56.66	56.98	56.94	56.90	57.28

Table 7: Average score of 6 few-shot tasks according to change in the value of k.

performance. While fine-tuning can boost task-specific accuracy, quantization-aware PEFT must overcome the accuracy degradation from quantization. We compared two quantization schemes, OWQ and round-to-nearest (RTN) method on Llama-2 7B. OWQ searches clipping value and reduces quantization error by reconstruction, while RTN uses naive minmax quantization. Tab. 6 shows that few-shot accuracies using OWQ as quantization are consistently better than RTN. Interestingly, perplexity score after quantization (without fine-tuning) is similar in both cases. Analyzing more on the effect of quantization on fine-tuning is our future research direction.

C.3 The Number of Weak Columns k

In this paper, k presents the number of weak columns, which are preserved in full-precision in each linear weight. There is a trade-off depending on the k value between the number of tunable parameters and the overall model cost. To find out the effect of k on tuning performance, we measured the score by changing the value of k (Tab. 7). When we used the same learning rate, the score seemed irrelevant to the k values. The result is the average of the three seed values, and although there is some variation, it was confirmed that the performance improves as the k value increases. $k = 16$ and 128 were used for the experiments in Tab. 2 to match the number of learnable parameters to the control group.

	QEFT		OWQ	
	Time(m)	Storage(MB)	Time(m)	Storage(MB)
7B	39.5	4107	25.6	3923
13B	70.0	7561	46.7	7200
70B	107.0	37260	246.7	35296

Table 8: Quantization costs for QEFT and OWQ.

C.4 Quantization Cost for QEFT and OWQ

We measured quantization time and storage cost for QEFT and OWQ on Llama-2 family (Tab. 8). In the 7B and 13B models, QEFT is more expensive than OWQ because it applies group-wise quantization and grid search. However, please note that the quantization process time is one-time cost and can be amortized. If we create the quantized weights for each base model once, they can be used for multiple QEFT fine-tuning for several datasets and tasks. For the Llama-2 70B model, OWQ takes 4.1 hours to perform quantization and reconstruction on the single A100 80GB. On the other hand, group-wise quantization process of QEFT 70B takes about 1.8 hours because QEFT 70B utilizes minmax and round-to-nearest quantization instead of grid search and reconstruction, respectively. Please refer to Sec. 6.1 for more details about the quantization of the QEFT 70B case. Regarding the storage cost, grid search also does not affect the cost as it only alters the value of each quantization parameter (scales and zero points). Group-wise quantization slightly increases storage overhead due to the increased number of group-wise parameters. Tab. 8 shows the storage overhead of group-wise quantization for all model sizes (with k=128). We can see that group-wise quantization imposes insignificant storage overhead compared to the overall size of the model.

C.5 Measuring the Tuning Time

The tuning cost reported in Tab. 2 measures the time required to train 1 epoch of the platypus dataset. Please note that these training time results (except Platypus 70B) were measured during the actual process of training QEFT or while reproducing our control group results, under the same environment of a single A100 80GB memory. We referenced the tuning cost of Platypus 70B from its paper due to its resource-intensive nature (requiring 8xA100 GPUs) and lengthy training time.

Additionally, we compare the forward and backward times for a single model run with an input length of 512. These results are reported in Tab. 9.

LLaMa-2 Model	Forward	Backward	Total
QEFT 7B (k=128)	54ms	160ms	214ms
QLoRA 7B (r=64)	138ms	340ms	478ms
QEFT 13B (k=128)	90ms	290ms	380ms
QLoRA 13B (r=64)	254ms	590ms	844ms

Table 9: Speed measurement results for forward and backward operations in QEFT and QLoRA. Gradient checkpointing is applied to all methods.

	7B	13B	70B
LoRA	14.33	27.08	OOM
Platypus	8.16	14.77	OOM
QLoRA	5.16	9.12	41.46
QA-LoRA	5.01	9.03	42.98
OWQ	4.10	7.41	35.75
QEFT	4.24	7.69	37.25

Table 10: Training memory footprint (GB) of each PEFT method.

Compared to QLoRA, QEFT is significantly faster in both forward and backward passes because there is no low-rank decomposed path. In particular, the backward pass is further accelerated due to the reduced amount of computation, as explained in Sec. 3.3. When we compare the combined forward and backward time of QEFT and QLoRA, QEFT is more than twice as fast, which is consistent with the overall training time of the two methods in Tab. 2.

D Memory Footprint

D.1 Fine-tuning Memory Footprint

Tab. 10 shows the memory footprint during fine-tuning using each PEFT method. We used the same configuration as Tab. 2 for all methods, except input sequence length, where we used 512 for this. OWQ and QEFT use k=16.

First, the training memory footprint is dominated by the base model size. LoRA has the largest memory footprint because it uses a 16-bit base model, followed by Platypus with an 8-bit base model. Please note that as OWQ does not provide tuning code, we implemented it based on our setup. Therefore, OWQ also benefits from QEFT's customized backward implementation, while both memory footprints are much lower than other methods. The slight difference in memory footprint between OWQ and QEFT is due to the use of groups.

We further detailed a breakdown of memory usage during training in Fig. 8. The memory used by the input-enabled gradient in all methods is small because gradient checkpointing is applied and the

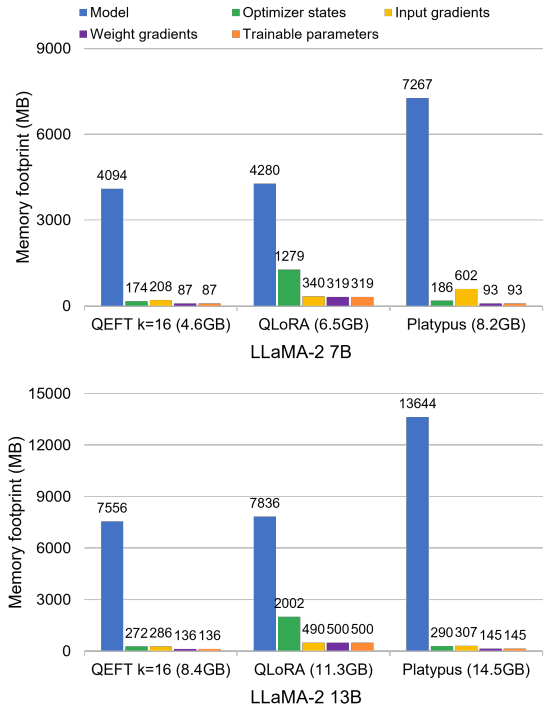


Figure 8: The memory usage for training with QEFT and baselines.

Llama-2	unoptimized (GB)	optimized (GB)
7B	14.40	**6.40**
13B	18.38	**8.38**
70B	32.51	**16.51**

Table 11: Peak GPU Memory Usage during quantization process of QEFT (and other OPTQ-based methods).

batch size is 1. At the k=16 setting, QEFT incurs the smallest memory among QEFT, QLoRA, and Platypus.

D.2 GPU Memory Overhead of the Quantization process

Additionally, we report the peak GPU memory usage during the QEFT quantization process in Tab. 11.

First, the memory usage of the quantization process in QEFT, OWQ, and QA-LoRA follows the block-wise reconstruction method of OPTQ, where only each decoder block is kept in GPU memory every time. Therefore, the memory consumption of these methods is almost identical. When we use OPTQ's implementation, the forward process of the calibration set between each block consumes the most GPU memory usage, as it keeps both 128 input and 128 output tensors on GPU memory. This requirement is identical for QEFT, OPTQ, OWQ, and QA-LoRA, demanding a reasonable amount of GPU memory, as seen in the "unoptimized" column of the Table.

However, in our experiment, we further optimized the original OPTQ reconstruction process to lower peak GPU memory usage. By allowing only one calibration sample to be allocated to GPU memory at a time on demand, without affecting the quantization process, we reduced memory consumption by half ("optimized" column of the Table) over the OPTQ code. Please note that all of OPTQ-related algorithms benefit from this optimization as well. In environments where GPU memory for parameter-efficient fine-tuning is available, the base model can be quantized without difficulty.