QFT: QUANTIZED FULL-PARAMETER TUNING OF LLMS WITH AFFORDABLE RESOURCES

Zhikai Li[1] , Xiaoxuan Liu[2] , Banghua Zhu[2] , Zhen Dong[2]✉, Qingyi Gu[1]✉, Kurt Keutzer[2]
[1]Institute of Automation, Chinese Academy of Sciences [2]University of California, Berkeley
{zhikai.li,qingyi.gu}@ia.ac.cn
{xiaoxuan_liu,banghua,zhendong,keutzer}@berkeley.edu

ABSTRACT

Large Language Models (LLMs) have showcased remarkable impacts across a wide spectrum of natural language processing tasks. Fine-tuning these pretrained models on downstream datasets provides further significant performance gains; however, this process typically requires a large number of expensive, high-end GPUs. Although there have been efforts focused on parameter-efficient fine-tuning, they cannot fully unlock the powerful potential of full-parameter fine-tuning. In this paper, we propose QFT, a Quantized Full-parameter Tuning framework for LLMs that quantizes and stores all training states, including weights, gradients, and optimizer states, in INT8 format to reduce training memory, thereby enabling full-parameter fine-tuning on existing GPUs at an affordable cost. To ensure training performance, we make two key efforts: i) for quantized gradients and optimizer states, we theoretically prove that the Lion optimizer, with its property of consistent update magnitudes, is highly robust to quantization; ii) and for quantized weights, we employ the hybrid feature quantizer, which identifies and protects a small subset of sparse critical features while quantizing the remaining dense features, thus ensuring accurate weight updates without FP32 backups. Moreover, to support backpropagation in the integer context, we develop a stack-based gradient flow scheme with O(1) complexity, forming a unified integer training pipeline. As a result, QFT reduces the model state memory to 21% of the standard solution while achieving comparable performance, e.g., tuning a LLaMA-7B model requires only <30GB of memory, making it feasible on a single A6000 GPU.

1 INTRODUCTION

Large Language Models (LLMs), with up to hundreds of billions of parameters, have left an indelible mark on the landscape of natural language processing tasks, showcasing their remarkable impacts across a diverse spectrum of applications and domains (Touvron et al., 2023a;b; Brown et al., 2020; Zhang et al., 2022). Fine-tuning these pre-trained models on downstream datasets enhances their ability to understand and perform specific tasks (Zhao et al., 2023). However, due to the enormous number of parameters, the fine-tuning process relies on massive and expensive GPU resources, resulting in extremely high costs.

Parameter-efficient fine-tuning (PEFT), which tunes only a subset of parameters, is considered a practical choice in low-resource settings (Ding et al., 2022; Hu et al., 2021; Li & Liang, 2021; Zhang et al., 2025). However, due to the limited representational capacity of the smaller parameter set, its performance often falls short of expectations (Lv et al., 2023; Zhao et al., 2024). Consequently, we focus on full-parameter fine-tuning, with an emphasis on exploring memory optimization strategies to make it feasible on cost-effective resources.

We examine the full spectrum of memory usage in full-parameter fine-tuning, which can be categorized into three components: model states, activation, and other temporary or unusable memory, as shown in Figure 1. Model states, which include weights, gradients, and optimizer states (e.g., momentum and variances in Adam (Kingma & Ba, 2015)), consume the majority of the memory (Rajbhandari et al., 2020). For instance, when employing the standard FP32 setting with Adam, the

✉ Corresponding authors.

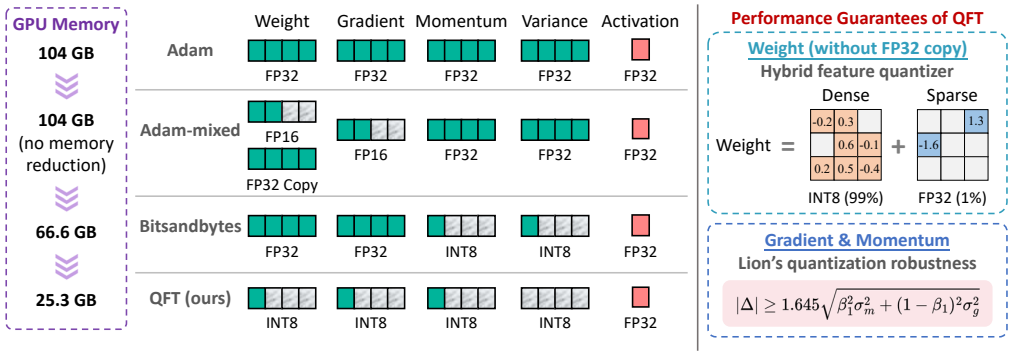


Figure 1: Comparison in GPU memory usage of different full-parameter fine-tuning methods. QFT significantly reduces training memory, enabling fine-tuning with affordable resources. To ensure the performance of quantized fine-tuning, QFT adopts the hybrid feature quantizer for weights, and for gradients and momentum, we theoretically prove that Lion exhibits high robustness to quantization, thereby ensuring comparable convergence to FP32 Adam.

memory allocation for weights, gradients, momentum, and variances amounts to 4 times the number of parameters. As a result, tuning a LLaMA-7B model necessitates a minimum of 104GB of RAM, which presents a formidable challenge given the limitations of current GPU capacities. Notably, although mixed-precision training (Micikevicius et al., 2017) reduces computation precision, it requires storing an additional FP32 weight copy, hence it only accelerates training but fails to address the memory consumption issues.

In this paper, we introduce QFT, a Quantized Full-parameter Tuning framework for training memory optimization. specifically, QFT quantizes and stores all training states in INT8 format, significantly reducing memory consumption and enabling full-parameter fine-tuning on existing GPUs at an affordable cost. In contrast to traditional quantization-aware training (QAT), QFT focuses specifically on memory optimization by *storing* all parameters in low-bit format. To maintain training performance, we make two key efforts: ❶ **For quantized gradients and optimizer states**, we theoretically prove that the Lion optimizer (Chen et al., 2023), which tracks only momentum and produces consistent update magnitudes, exhibits strong robustness to quantization. Thus, we employ the Lion optimizer to minimize the effects of quantization of gradients and momentum. ❷ **For quantized weights**, we employ the hybrid feature quantizer, which selectively retains a small set of sparse critical features while quantizing the majority of dense features, thereby ensuring accurate weight updates. It does not rely on FP32 backups, thus achieving better memory efficiency than mixed-precision training. On this basis, to enable integer-based backpropagation, we design a stack-based gradient flow scheme with O(1) complexity, constructing a unified integer training pipeline. It is also worth noting that QFT adopts the INT8 format by default, which is broadly supported by most hardware. This design avoids reliance on specialized data types like FP8 (Micikevicius et al., 2022) that require high-end GPUs, thus enabling full utilization of existing mid- and low-end GPUs. Our contributions can be summarized as follows:

- We propose QFT, a Quantized Full-parameter Tuning framework for LLMs. It achieves training memory optimization by reducing storage precision, enabling full-parameter fine-tuning on affordable resources, which separates it from traditional QAT. In addition, QFT offers strong compatibility and can be seamlessly integrated into mainstream LLM training tools.

- To ensure training performance, we first theoretically prove the robustness of the Lion optimizer to quantization, ensuring the reliability of *quantized gradients and optimizer states*. Then, we protect a small subset of critical features within the weights while quantizing the remaining dense features, effectively preserving the accuracy of *quantized weight updates*. Moreover, we develop a stack-based gradient flow scheme with O(1) complexity to enable integer backpropagation.

- We perform instruction tuning on the pre-trained LLaMA-2 models and extensively evaluate performance on various benchmarks. The results demonstrate that our QFT, with memory usage reduced to 21%, achieves comparable performance to standard floating-point training.

2 Related Work

Efficient Optimizer The Adam family optimizers are widely used for training Transformer models (Kingma & Ba, 2015; Loshchilov & Hutter, 2017), but their optimizer states (momentum and variance) introduce substantial memory overhead. To mitigate this, several memory-efficient alternatives have been proposed. LOMO (Lv et al., 2023) uses vanilla SGD, but suffers from slow convergence and poor stability (Li et al., 2023). BAdam (Luo et al., 2024) adopts a block coordinate descent framework with Adam-style updates. Adafactor (Shazeer & Stern, 2018) reduces memory by storing only aggregated statistics, but remains prone to instability. In this work, we adopt the Lion optimizer (Chen et al., 2023), which tracks only momentum while achieving performance comparable to Adam. More importantly, its sign-based update yields consistent update magnitudes, making it particularly suitable for robust quantization of gradients and optimizer states.

Quantization for Memory Optimization Most existing quantization methods focus on inference efficiency (Gholami et al., 2022; Dong et al., 2019; 2020; Kim et al., 2023; Li et al., 2022a;b; Li & Gu, 2022; Jacob et al., 2018). This line of work differs fundamentally from traditional QAT (Jacob et al., 2018; Liu et al., 2023), which inserts fake quantization nodes during training while keeping parameter storage and computation in floating point, and thus does not reduce training memory usage. In contrast, quantization-based memory optimization aims to store parameters in low precision to directly reduce training memory. For example, Bitsandbytes (Dettmers et al., 2021) applies block-wise quantization to compress optimizer states, and QLoRA (Dettmers et al., 2023) stores frozen pre-trained weights in quantized form while keeping adapters in floating point. In this work, we focus on full-parameter fine-tuning and quantize all model states to achieve comprehensive memory savings without sacrificing fine-tuning performance.

Other Memory Optimization Methods Previous work has focused on reducing activation memory, including activation offloading (Huang et al., 2020; Wang et al., 2018; Peng et al., 2020) and gradient checkpointing (Chen et al., 2016; Kumar et al., 2019; Jain et al., 2020; Kirisame et al., 2020). Activation offloading moves activations to external memory at the cost of data transfer, while gradient checkpointing discards activations in the forward pass and recomputes them during backpropagation. There are also customized schemes for other training states. GaLore (Zhao et al., 2024) reduces gradient memory via low-rank projection, and LOMO (Lv et al., 2023) fuses gradient computation and parameter update, reducing gradient memory to $O(1)$. However, LOMO is incompatible with gradient accumulation for large batch sizes, limiting it to unstable small-batch training. In contrast, our framework is orthogonal to and compatible with all these methods.

3 Methodology

3.1 Lion Optimizer: Robust Quantization of Gradients and Momentum

In contrast to Bitsandbytes (Dettmers et al., 2021) which adopts advanced quantization strategies, we prioritize ease of use by applying the simplest uniform quantizer to gradients and momentum, defined as follows:

$$Quant: \boldsymbol{X}^{(\mathbb{Z})} = \text{clip}\left(\left\lfloor \frac{\boldsymbol{X}}{s} \right\rceil + z, 0, 2^b - 1\right), \quad De\text{-}quant: \hat{\boldsymbol{X}} = s\left(\boldsymbol{X}^{(\mathbb{Z})} - z\right) \approx \boldsymbol{X}, \tag{1}$$

where $\boldsymbol{X}$ is the floating-point vector, $\boldsymbol{X}^{(\mathbb{Z})}$ is the quantized integer vector, $\lfloor \cdot \rceil$ denotes the round function, and $b \in \mathbb{N}$ is the quantization bit-width. $s \in \mathbb{R}^+$ and $z \in \mathbb{Z}$ are the quantization scale and zero-point, which are determined by the arithmetic lower and upper bounds of $\boldsymbol{X}$ as follows:

$$s = \frac{\max(\boldsymbol{X}) - \min(\boldsymbol{X})}{2^b - 1}, \quad z = \left\lfloor -\frac{\min(\boldsymbol{X})}{s} \right\rceil. \tag{2}$$

The simplicity of the quantizer effectively ensures computational efficiency but also raises concerns about potential impacts on training performance. Fortunately, Lion (Chen et al., 2023) (detailed in Appendix B), which tracks only momentum and applies updates of consistent magnitude to each parameter via the sign operation, can significantly mitigate the adverse effects of quantization. More specifically, Lion has inherent advantages for quantized fine-tuning as follows:

- **Simplicity:** Lion only keeps track of the momentum, which saves memory by avoiding storing variances, while eliminating the potential effect of quantized variances.

- **Consistent Update Magnitudes:** Lion ensures that updates have the same magnitude for each parameter, which is determined through the sign operation. In a quantized setting, this consistency can mitigate potential imbalances or inaccuracies introduced by limited precision.

For the consistent update magnitude property, we provide a detailed proof below to demonstrate Lion's high robustness to quantization of gradients and momentum.

Assumption 1. *Assume that:*

- *(Additive quantization error) The quantization error is additive, i.e.,* $\hat{x} = x + \delta_x$.
- *(Bounded Gaussian error) The quantization error* δ_x *follows a Gaussian distribution* $\mathcal{N}(0, \sigma_x^2)$*, where* σ_x^2 *is small and bounded.*

Lemma 1. *Under Assumption 1, when quantizing gradients and momentum in Lion, if the increment* Δ *satisfies* $|\Delta| \geq 1.645\sqrt{\beta_1^2\sigma_m^2 + (1-\beta_1)^2\sigma_g^2}$*, then with 95% probability,* $\operatorname{sign}(\Delta)$ *remains invariant under quantization.*

Proof. Given Assumption 1, we have:

$$\hat{m} = m + \delta_m,\ \hat{g} = g + \delta_g, \quad \text{where } \delta_m \sim \mathcal{N}(0, \sigma_m^2),\ \delta_g \sim \mathcal{N}(0, \sigma_g^2).$$

When quantizing gradients and momentum, the increment becomes:

$$\Delta' = \beta_1\hat{m} + (1-\beta_1)\hat{g} = \Delta + \beta_1\delta_m + (1-\beta_1)\delta_g.$$

Let $\delta_\Delta = \beta_1\delta_m + (1-\beta_1)\delta_g$. Since δ_m and δ_g are independent Gaussian distributions, based on homogeneity and additivity, δ_Δ follows a Gaussian distribution:

$$\delta_\Delta \sim \mathcal{N}(0, \sigma_\delta^2) := \mathcal{N}(0, \beta_1^2\sigma_m^2 + (1-\beta_1)^2\sigma_g^2).$$

To ensure that the sign of Δ remains invariant after quantization, we require that $\operatorname{sign}(\hat{\Delta}) = \operatorname{sign}(\Delta)$, i.e., Δ and $\Delta + \delta_\Delta$ must have the same sign. First, consider the case $\Delta > 0$, we require $\Delta + \delta_\Delta > 0 \Leftrightarrow \delta_\Delta > -\Delta$, thus the probability of sign preservation is:

$$P(\delta_\Delta > -\Delta) = 1 - P(\delta_\Delta \leq -\Delta).$$

Standardizing δ_Δ yields:

$$P(\delta_\Delta \leq -\Delta) = P\left(\frac{\delta_\Delta}{\sigma_\delta} \leq -\frac{\Delta}{\sigma_\delta}\right) = \Phi\left(-\frac{\Delta}{\sigma_\delta}\right),$$

where $\Phi(\cdot)$ denotes the cumulative distribution function of the standard normal distribution. To ensure the probability of sign flip is at most 5%, we require $\Phi\left(-\frac{\Delta}{\sigma_\delta}\right) \leq 0.05$. Since $\Phi^{-1}(0.05) = -1.645$, it follows that $-\frac{\Delta}{\sigma_\delta} \leq -1.645 \Leftrightarrow \Delta \geq 1.645\sigma_\delta$. Substituting σ_δ gives:

$$\Delta \geq 1.645\sqrt{\beta_1^2\sigma_m^2 + (1-\beta_1)^2\sigma_g^2}.$$

Similarly, when both Δ and $(\Delta + \delta_\Delta)$ are negative, we have $\Delta \leq -1.645\sqrt{\beta_1^2\sigma_m^2 + (1-\beta_1)^2\sigma_g^2}$.

Therefore, in summary, if $|\Delta| \geq 1.645\sqrt{\beta_1^2\sigma_m^2 + (1-\beta_1)^2\sigma_g^2}$, then with at least 95% probability, $\operatorname{sign}(\Delta)$ remains invariant under quantization, i.e., $\operatorname{sign}(\hat{\Delta}) = \operatorname{sign}(\Delta)$. □

Remark 1. *We empirically observe the ratio* $\frac{|\Delta|}{\sigma_\delta}$ *across different iterations and layers. Our results indicate that in at least 97.9% of cases, the ratio exceeds 1.645, thereby satisfying the condition stated above. Further details are provided in Appendix C.*

3.2 Hybrid Feature Quantizer: Accurate Updates of Quantized Weights

In addition to gradients and momentum, we also store weights in integer format for memory efficiency. This quantized storage is designed purely for training memory optimization and is fundamentally different from traditional QAT (Jacob et al., 2018), which inserts fake quantization nodes.

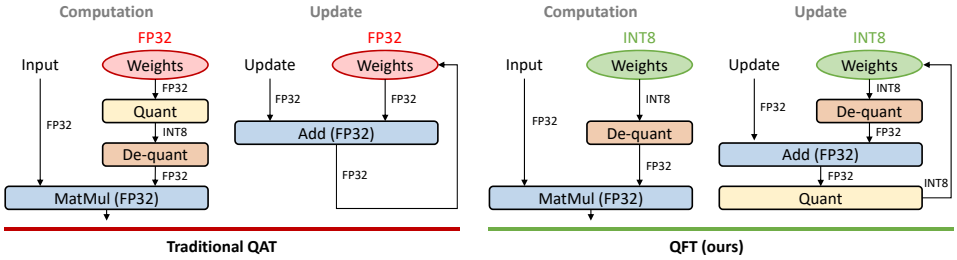


Figure 2: Comparison between our QFT and traditional QAT in the computation and update procedures of weights. QAT stores the weights in the floating-point format and adds fake quantization nodes to the computation. Conversely, in our QFT, the weights are stored in the low-precision integer format, which are de-quantized on-the-fly into the floating-point format for computation, resulting in a significant reduction in memory usage.

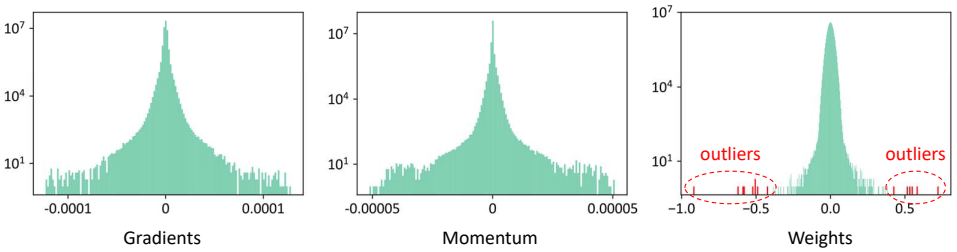


Figure 3: Illustration of the model state distributions when training a LLaMA-2-7B model. The weight values are from the final down projection layer, and the gradient and momentum values are fetched on the 200th training step. The gradients and momentum show a canonical centralized distribution with few outliers, while the range of the weights increases by three orders of magnitude and exhibits extreme outliers, posing a significant challenge.

In our framework, weights are stored as quantized integers but are de-quantized to floating-point for computation, trading a small amount of extra computation for substantial memory savings. A comparison with QAT is shown in Figure 2.

However, weight quantization is considerably more challenging. By analyzing weight distributions, we make two key observations: (i) a small number of outliers play a crucial role in representation and typically correspond to critical features (Kim et al., 2023; Frantar et al., 2022; Lin et al., 2023); (ii) these outliers significantly hinder quantization by greatly expanding the dynamic range of weights, which is about three orders of magnitude larger than that of momentum, as shown in Figure 3. These conflicting properties make outliers particularly difficult to handle.

Fortunately, the sparsity of outliers provides an opportunity. Statistics show that 99% of the values lie within only 20% of the full range. This motivates us to adopt a hybrid feature quantizer inspired by SqueezeLLM (Kim et al., 2023), which preserves the top 1% sparse but critical features and compacts the remaining dense distribution. Formally, the method is defined as follows:

$$\begin{aligned} \boldsymbol{W} &= \boldsymbol{D} + \boldsymbol{S}, \\ \text{s.t.}\quad \boldsymbol{D} &= \boldsymbol{W} \odot \mathbb{I}\left(T_{\min} \leq \boldsymbol{W} \leq T_{\max}\right), \\ \boldsymbol{S} &= \boldsymbol{W} \odot \mathbb{I}\left(\boldsymbol{W} < T_{\min} \text{ or } \boldsymbol{W} > T_{\max}\right), \end{aligned} \tag{3}$$

where $\boldsymbol{D}$ is a dense matrix containing the central values and $\boldsymbol{S}$ is a sparse matrix capturing the outliers. $T_{\min}$ and $T_{\max}$ denote outlier thresholds, and $\mathbb{I}(\cdot)$ is the indicator function. This decomposition is numerically simple and highly efficient, introducing negligible training overhead.

The dense matrix is then quantized using the uniform quantizer in Equation 1, while the sparse matrix keeps its values in floating point. Since the outliers account for only a small fraction (1%), the sparse matrix can be stored in memory-efficient formats such as CSR, incurring negligible memory overhead while preserving critical information.

Moreover, QFT has a key advantage over mixed-precision training, which requires maintaining an FP32 copy of the weights. In mixed-precision FP16 training, forward and backward passes use FP16 weights and gradients, but updates must be applied to an FP32 master copy due to the limited

precision and overflow risks of FP16. In contrast, QFT does not rely on simple numerical truncation: it preserves sparse critical features and uniformly quantizes the dense values into integers. As a result, QFT enables stable training without maintaining an additional FP32 weight copy.

3.3 The Integer Training Pipeline

In this section, we integrate the above strategies into a memory-efficient fine-tuning framework for LLMs. We describe each training stage in detail, including forward propagation, backward propagation, and parameter updates, with particular emphasis on the stack-based gradient flow scheme with $O(1)$ complexity in the integer setting.

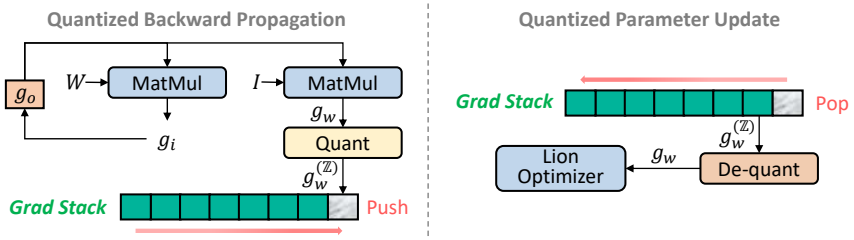


Figure 4: The proposed stack-based gradient flow scheme, which enables storage and $O(1)$ complexity access to integer gradients. This effectively eliminates AutoGrad's dependency on floating-point formats, enabling efficient gradient propagation in the context of integer weights.

Quantized Forward Propagation In our framework, weights are stored in quantized integer format to reduce memory usage. During forward propagation, these low-precision weights are de-quantized to floating point on the fly, enabling high-precision computation, as illustrated in Figure 2.

Algorithm 1 Gradient Flow of Quantized Weights

```
# T_l : saved tensors in forward pass of layer l
# g_o : gradient of the current layer's output

S_g ← stack()
for l = L, L − 1, ··· , 1 do
    I_l, W_l^(ℤ) ← T_l
    W_l ← dequant(W_l^(ℤ))
    calculate gradients of I_l and W_l
    g_i ← matmul(g_o, W_l)
    g_w ← matmul(g_o^T, I_l)
    g_w^(ℤ) ← quant(g_w)                ▷ store as INT8
    push(S_g, g_w^(ℤ))                  ▷ collect gradient
    assign g_o of layer (l-1)
    g_o ← g_i
end for
```

Algorithm 2 Quantized Lion Optimizer

```
# β_1, β_2, λ, η, f : optimizer parameters
# m_l : optimizer momentum of layer l

for l = 1, 2, ··· , L do
    g_w^(ℤ) ← pop(S_g)                  ▷ retrieve gradient
    g_w ← dequant(g_w^(ℤ))
    m_l ← dequant(m_l^(ℤ))
    W_l ← dequant(W_l^(ℤ))
    update model parameters
    Δ ← β_1 m_l + (1 − β_1) g_w
    W_l ← W_l − η(sign(Δ) + λ W_l)
    update EMA of g_w
    m_l ← β_2 m_l + (1 − β_2) g_w
    m_l^(ℤ) ← quant(m_l)                ▷ store as INT8
    W_l^(ℤ) ← quant(W_l)                ▷ store as INT8
end for
```

Quantized Backward Propagation During backward propagation, the loss is propagated layer by layer and gradients are computed for each parameter. These gradients must be stored in memory to guide subsequent updates. However, in mainstream frameworks such as PyTorch, only floating-point parameters can hold gradients. As a result, gradients for integer-stored weights cannot be computed or stored directly using the standard automatic differentiation (AutoGrad) mechanism.

To address this issue, we design a new gradient flow for integer-stored weights (Figure 4, Algorithm 1). As in the forward pass, weights are first de-quantized to floating point, and gradients w.r.t. inputs and weights are computed via the chain rule. To handle gradient storage, we further propose a global stack-based retention scheme, where each layer's weight gradient is pushed onto the stack during backpropagation in reverse forward order.

Quantized Parameter Update Finally, parameters are updated using the standard Lion optimizer, except that both gradients and momentum are stored in integer format (Algorithm 2). Gradients are retrieved by popping from the global stack in $O(1)$ time. This efficiency stems from our gradient

Table 1: Memory usage (in GB) when fine-tuning the LLaMA-2-7B model using different methods. We report the memory profiles, total allocated memory, and peak allocated memory.

Method	Weights	Grads	Optimizer States			Activation	Total	Peak
			Weight Copies	Momentum	Variances			
Adam-FP32	25.1	25.1	-	25.1	25.1	3.75	104	129
Adam-FP16 mixed	12.6	12.6	25.1	25.1	25.1	3.75	104	123
Bitsandbytes	25.1	25.1	-	6.31	6.31	3.75	66.6	86.6
Lion-FP32	25.1	25.1	-	25.1	-	3.75	79.1	101
QFT (ours)	7.42	7.06	-	7.06	-	3.75	25.3	28.9

Table 2: Few-shot performance of different models on various standard benchmarks. Here, the number of shots is aligned to Open LLM Leaderboard (HuggingFace, 2023a).

Pre-trained	Tuning	Full-param	ARC-c (25-shot)	HellaSwag (10-shot)	MMLU (5-shot)	TruthfulQA-mc (0-shot)	Average
LLaMA-2-7B	-	-	53.1	78.6	46.9	38.8	54.4
	LoRA	×	53.0	78.0	47.8	45.8	56.2
	FT-Adam	✓	53.6	77.3	49.4	51.5	58.0
	FT-Lion	✓	53.5	77.6	49.2	51.3	57.9
	FT-Bnb	✓	53.1	76.9	49.0	51.1	57.5
	QFT (ours)	✓	52.9	76.7	48.8	51.1	57.4
LLaMA-2-13B	-	-	59.4	82.1	55.8	37.4	58.7
	LoRA	×	57.3	81.2	55.6	44.7	59.7
	FT-Adam	✓	57.0	81.2	55.8	50.9	61.2
	FT-Lion	✓	56.6	81.2	55.6	50.7	61.0
	FT-Bnb	✓	56.0	80.9	55.4	49.4	60.4
	QFT (ours)	✓	56.2	81.0	55.9	48.6	60.4

flow design: gradients are pushed onto the stack during backpropagation from the last layer to the first, and popped during optimization in the reverse order, so the current layer's gradient is always on top of the stack, fully exploiting the LIFO property.

4 Experiments

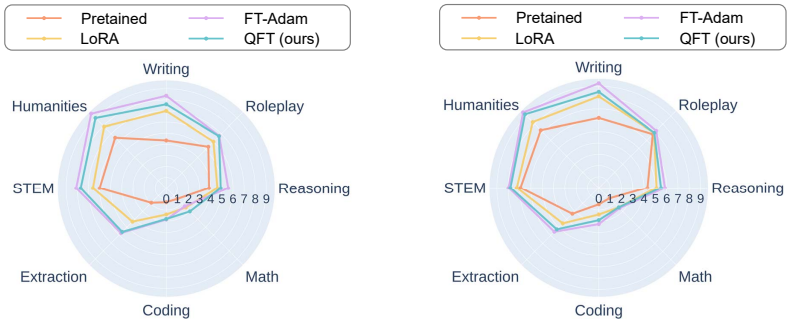
4.1 Experimental Setup

Baseline Methods We evaluate QFT in terms of both training memory and performance. For training memory, QFT is compared to standard FP32 Adam (Kingma & Ba, 2015), mixed-precision FP16 Adam (Micikevicius et al., 2017), BitsandBytes (Dettmers et al., 2021), and standard FP32 Lion (Chen et al., 2023). For training performance, we consider four baselines: **LoRA**, which is a PEFT method (Hu et al., 2021); **FT-Adam**, which performs full-parameter fine-tuning using FP32 Adam (Kingma & Ba, 2015); **FT-Lion**, which performs fine-tuning using FP32 Lion (Chen et al., 2023); **FT-Bnb**, which uses Bitsandbytes (Dettmers et al., 2021) with quantized optimizer states.

Models, benchmarks, datasets, and training details are presented in Appendix A.

4.2 Memory Profile

The memory usage of different methods are reported in Table 1. All experiments employ gradient checkpointing by default to reduce activation memory. Under standard Adam , each model state consumes 25.1GB of RAM. This issue persists in mixed-precision settings. Lion reduces memory usage by 25% by tracking only momentum. BitsandBytes further compresses optimizer states via quantization, saving 37GB. Our QFT applies full quantization to all model states, requiring only 21.5GB of GPU memory, which is just 21% of that used by standard Adam.



(a) 7B models (b) 13B models

Figure 5: Radar charts of each capability in MT-Bench of different tuning method.

4.3 Performance Evaluation

Few-Shot Evaluation We perform few-shot performance evaluation across a range of established benchmarks, with results presented in Table 2. On LLaMA-2-7B, QFT boosts the average performance score from 54.4 to 57.4, which comes within 0.6 of full-precision fine-tuning with Adam (FT-Adam), while outperforming LoRA. We also provide a qualitative analysis in Appendix E.

MT-Bench Score We further adopt MT-Bench, an advanced benchmark designed to evaluate the conversational capabilities of LLMs. It comprises a series of challenging multi-turn, open-ended questions that closely reflect human conversational preferences, with GPT-4 serving as the automatic evaluator. The results are summarized in Table 3. For 7B models, LLaMA-2 achieves a modest score of 3.83. While LoRA yields a slight improvement, it still falls short by 0.97 compared to full-parameter fine-tuning. Notably, QFT closes this gap, achieving performance comparable to FT-Adam.

Table 3: MT-Bench scores using GPT-4, which can reflect model's conversational abilities.

Pre-trained	Tuning	Full-param	MT-Bench Score
LLaMA-2-7B (Score: 3.83)	LoRA	×	5.11
	FT-Adam	✓	6.08
	FT-Lion	✓	6.11
	FT-Bnb	✓	5.94
	QFT (ours)	✓	5.95
LLaMA-2-13B (Score: 4.69)	LoRA	×	5.74
	FT-Adam	✓	6.46
	FT-Lion	✓	6.44
	FT-Bnb	✓	6.30
	QFT (ours)	✓	6.27

We also provide radar charts in Figure 5. It demonstrates that QFT delivers comprehensive and consistent improvements across all evaluated metrics compared to the pre-trained LLaMA-2 baseline.

Training Throughput and Convergence We measure the average time per 1000 training steps. Due to quant-dequant overhead, QFT incurs a 1.2-1.3× increase in training time compared to LLaMA-2-FT with FP32 Adam. This time–memory trade-off is considered significant, especially under constrained memory budgets. Moreover, Figure 6 compares the training losses of QFT and LLaMA-2-FT, showing that QFT achieves comparable convergence.

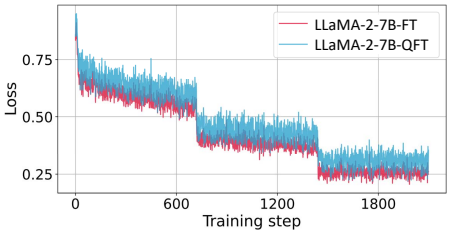


Figure 6: Comparison of training loss curves.

5 Conclusion

In this paper, we propose QFT, a memory-efficient framework for full-parameter fine-tuning of LLMs under quantized training. QFT quantizes all training states, including weights, gradients, and optimizer states, into INT8, enabling end-to-end training with substantially reduced memory usage while preserving performance. To ensure training stability, we introduce two key techniques: a quantization-robust optimizer (Lion) for momentum and gradients, and a hybrid feature quantizer that preserves sparse critical weight features. We further propose a stack-based gradient flow with $O(1)$ complexity to support efficient integer-domain training. Experiments show that QFT reduces model state memory to 21% of standard FP32 fine-tuning, enabling LLaMA-2-7B to be fine-tuned on commodity GPUs such as an A6000 with less than 30GB memory.

REFERENCES

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

Tianqi Chen, Bing Xu, Chiyuan Zhang, and Carlos Guestrin. Training deep nets with sublinear memory cost. *arXiv preprint arXiv:1604.06174*, 2016.

X Chen, C Liang, D Huang, E Real, K Wang, Y Liu, H Pham, X Dong, T Luong, CJ Hsieh, et al. Symbolic discovery of optimization algorithms. arxiv 2023. *arXiv preprint arXiv:2302.06675*, 2023.

Wei-Lin Chiang, Zhuohan Li, Zi Lin, Ying Sheng, Zhanghao Wu, Hao Zhang, Lianmin Zheng, Siyuan Zhuang, Yonghao Zhuang, Joseph E. Gonzalez, Ion Stoica, and Eric P. Xing. Vicuna: An open-source chatbot impressing gpt-4 with 90%* chatgpt quality, March 2023. URL https://lmsys.org/blog/2023-03-30-vicuna/.

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

Tim Dettmers, Mike Lewis, Sam Shleifer, and Luke Zettlemoyer. 8-bit optimizers via block-wise quantization. In *International Conference on Learning Representations*, 2021.

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

Ning Ding, Yujia Qin, Guang Yang, Fuchao Wei, Zonghan Yang, Yusheng Su, Shengding Hu, Yulin Chen, Chi-Min Chan, Weize Chen, et al. Delta tuning: A comprehensive study of parameter efficient methods for pre-trained language models. *arXiv preprint arXiv:2203.06904*, 2022.

Zhen Dong, Zhewei Yao, Amir Gholami, Michael W Mahoney, and Kurt Keutzer. Hawq: Hessian aware quantization of neural networks with mixed-precision. In *Proceedings of the IEEE/CVF International Conference on Computer Vision*, pp. 293–302, 2019.

Zhen Dong, Zhewei Yao, Daiyaan Arfeen, Amir Gholami, Michael W Mahoney, and Kurt Keutzer. Hawq-v2: Hessian aware trace-weighted quantization of neural networks. *Advances in neural information processing systems*, 33:18518–18529, 2020.

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

Leo Gao, Jonathan Tow, Stella Biderman, Sid Black, Anthony DiPofi, Charles Foster, Laurence Golding, Jeffrey Hsu, Kyle McDonell, Niklas Muennighoff, Jason Phang, Laria Reynolds, Eric Tang, Anish Thite, Ben Wang, Kevin Wang, and Andy Zou. A framework for few-shot language model evaluation, September 2021. URL https://doi.org/10.5281/zenodo.5371628.

Amir Gholami, Sehoon Kim, Zhen Dong, Zhewei Yao, Michael W Mahoney, and Kurt Keutzer. A survey of quantization methods for efficient neural network inference. In *Low-Power Computer Vision*, pp. 291–326. Chapman and Hall/CRC, 2022.

Dan Hendrycks, Collin Burns, Steven Basart, Andy Zou, Mantas Mazeika, Dawn Song, and Jacob Steinhardt. Measuring massive multitask language understanding. *arXiv preprint arXiv:2009.03300*, 2020.

Edward J Hu, Yelong Shen, Phillip Wallis, Zeyuan Allen-Zhu, Yuanzhi Li, Shean Wang, Lu Wang, and Weizhu Chen. Lora: Low-rank adaptation of large language models. *arXiv preprint arXiv:2106.09685*, 2021.

Chien-Chin Huang, Gu Jin, and Jinyang Li. Swapadvisor: Pushing deep learning beyond the gpu memory limit via smart swapping. In *Proceedings of the Twenty-Fifth International Conference on Architectural Support for Programming Languages and Operating Systems*, pp. 1341–1355, 2020.

HuggingFace. Open llm leaderboard, 2023a. URL https://huggingface.co/spaces/HuggingFaceH4/open_llm_leaderboard.

HuggingFace. Sharegpt data, 2023b. URL https://huggingface.co/datasets/anon8231489123/ShareGPT_Vicuna_unfiltered?doi=true.

Benoit Jacob, Skirmantas Kligys, Bo Chen, Menglong Zhu, Matthew Tang, Andrew Howard, Hartwig Adam, and Dmitry Kalenichenko. Quantization and training of neural networks for efficient integer-arithmetic-only inference. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pp. 2704–2713, 2018.

Paras Jain, Ajay Jain, Aniruddha Nrusimha, Amir Gholami, Pieter Abbeel, Joseph Gonzalez, Kurt Keutzer, and Ion Stoica. Checkmate: Breaking the memory wall with optimal tensor rematerialization. *Proceedings of Machine Learning and Systems*, 2:497–511, 2020.

Sehoon Kim, Coleman Hooper, Amir Gholami, Zhen Dong, Xiuyu Li, Sheng Shen, Michael W Mahoney, and Kurt Keutzer. Squeezellm: Dense-and-sparse quantization. *arXiv preprint arXiv:2306.07629*, 2023.

Diederik P Kingma and Jimmy Ba. Adam: A method for stochastic optimization. In *International Conference on Learning Representations*, 2015.

Marisa Kirisame, Steven Lyubomirsky, Altan Haan, Jennifer Brennan, Mike He, Jared Roesch, Tianqi Chen, and Zachary Tatlock. Dynamic tensor rematerialization. *arXiv preprint arXiv:2006.09616*, 2020.

Ravi Kumar, Manish Purohit, Zoya Svitkina, Erik Vee, and Joshua Wang. Efficient rematerialization for deep networks. *Advances in Neural Information Processing Systems*, 32, 2019.

Bingrui Li, Jianfei Chen, and Jun Zhu. Memory efficient optimizers with 4-bit states. *arXiv preprint arXiv:2309.01507*, 2023.

Xiang Lisa Li and Percy Liang. Prefix-tuning: Optimizing continuous prompts for generation. *arXiv preprint arXiv:2101.00190*, 2021.

Zhikai Li and Qingyi Gu. I-vit: integer-only quantization for efficient vision transformer inference. *arXiv preprint arXiv:2207.01405*, 2022.

Zhikai Li, Liping Ma, Mengjuan Chen, Junrui Xiao, and Qingyi Gu. Patch similarity aware data-free quantization for vision transformers. In *European Conference on Computer Vision*, pp. 154–170. Springer, 2022a.

Zhikai Li, Junrui Xiao, Lianwei Yang, and Qingyi Gu. Repq-vit: Scale reparameterization for post-training quantization of vision transformers. *arXiv preprint arXiv:2212.08254*, 2022b.

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

Stephanie Lin, Jacob Hilton, and Owain Evans. Truthfulqa: Measuring how models mimic human falsehoods. *arXiv preprint arXiv:2109.07958*, 2021.

Zechun Liu, Barlas Oguz, Changsheng Zhao, Ernie Chang, Pierre Stock, Yashar Mehdad, Yangyang Shi, Raghuraman Krishnamoorthi, and Vikas Chandra. Llm-qat: Data-free quantization aware training for large language models. *arXiv preprint arXiv:2305.17888*, 2023.

Ilya Loshchilov and Frank Hutter. Decoupled weight decay regularization. *arXiv preprint arXiv:1711.05101*, 2017.

Qijun Luo, Hengxu Yu, and Xiao Li. Badam: A memory efficient full parameter optimization method for large language models. *Advances in Neural Information Processing Systems*, 37: 24926–24958, 2024.

Kai Lv, Yuqing Yang, Tengxiao Liu, Qinghui Gao, Qipeng Guo, and Xipeng Qiu. Full parameter fine-tuning for large language models with limited resources. *arXiv preprint arXiv:2306.09782*, 2023.

Paulius Micikevicius, Sharan Narang, Jonah Alben, Gregory Diamos, Erich Elsen, David Garcia, Boris Ginsburg, Michael Houston, Oleksii Kuchaiev, Ganesh Venkatesh, et al. Mixed precision training. *arXiv preprint arXiv:1710.03740*, 2017.

Paulius Micikevicius, Dusan Stosic, Neil Burgess, Marius Cornea, Pradeep Dubey, Richard Grisenthwaite, Sangwon Ha, Alexander Heinecke, Patrick Judd, John Kamalu, et al. Fp8 formats for deep learning. *arXiv preprint arXiv:2209.05433*, 2022.

Xuan Peng, Xuanhua Shi, Hulin Dai, Hai Jin, Weiliang Ma, Qian Xiong, Fan Yang, and Xuehai Qian. Capuchin: Tensor-based gpu memory management for deep learning. In *Proceedings of the Twenty-Fifth International Conference on Architectural Support for Programming Languages and Operating Systems*, pp. 891–905, 2020.

Samyam Rajbhandari, Jeff Rasley, Olatunji Ruwase, and Yuxiong He. Zero: Memory optimizations toward training trillion parameter models. In *SC20: International Conference for High Performance Computing, Networking, Storage and Analysis*, pp. 1–16. IEEE, 2020.

shareGPT. Sharegpt, 2023. URL https://sharegpt.com/.

Noam Shazeer and Mitchell Stern. Adafactor: Adaptive learning rates with sublinear memory cost. In *International Conference on Machine Learning*, pp. 4596–4604. PMLR, 2018.

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

Hugo Touvron, Louis Martin, Kevin Stone, Peter Albert, Amjad Almahairi, Yasmine Babaei, Nikolay Bashlykov, Soumya Batra, Prajjwal Bhargava, Shruti Bhosale, et al. Llama 2: Open foundation and fine-tuned chat models. *arXiv preprint arXiv:2307.09288*, 2023b.

Linnan Wang, Jinmian Ye, Yiyang Zhao, Wei Wu, Ang Li, Shuaiwen Leon Song, Zenglin Xu, and Tim Kraska. Superneurons: Dynamic gpu memory management for training deep neural networks. In *Proceedings of the 23rd ACM SIGPLAN symposium on principles and practice of parallel programming*, pp. 41–53, 2018.

Rowan Zellers, Ari Holtzman, Yonatan Bisk, Ali Farhadi, and Yejin Choi. Hellaswag: Can a machine really finish your sentence? *arXiv preprint arXiv:1905.07830*, 2019.

Jun Zhang, Jue WANG, Huan Li, Lidan Shou, Ke Chen, Yang You, Guiming Xie, Xuejian Gong, and Kunlong Zhou. Train small, infer large: Memory-efficient loRA training for large language models. In *The Thirteenth International Conference on Learning Representations*, 2025.

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

Jiawei Zhao, Zhenyu Zhang, Beidi Chen, Zhangyang Wang, Anima Anandkumar, and Yuandong Tian. Galore: Memory-efficient llm training by gradient low-rank projection. *arXiv preprint arXiv:2403.03507*, 2024.

Wayne Xin Zhao, Kun Zhou, Junyi Li, Tianyi Tang, Xiaolei Wang, Yupeng Hou, Yingqian Min, Beichen Zhang, Junjie Zhang, Zican Dong, et al. A survey of large language models. *arXiv preprint arXiv:2303.18223*, 2023.

Lianmin Zheng, Wei-Lin Chiang, Ying Sheng, Siyuan Zhuang, Zhanghao Wu, Yonghao Zhuang, Zi Lin, Zhuohan Li, Dacheng Li, Eric Xing, et al. Judging llm-as-a-judge with mt-bench and chatbot arena. *arXiv preprint arXiv:2306.05685*, 2023.

A Experimental Setup

Models and Benchmarks We conduct evaluation of the proposed QFT by fine-tuning LLaMA-2 (Touvron et al., 2023b), including the 7B and 13B versions. The few-shot performance of fine-tuned models is evaluated on various standard benchmarks, including ARC (Clark et al., 2018), HellaSwag (Zellers et al., 2019), MMLU (Hendrycks et al., 2020), and TruthfulQA (Lin et al., 2021). All results are obtained using the Language Model Evaluation Harness tool (Gao et al., 2021). In addition, we also use MT-Bench (Zheng et al., 2023) with GPT-4 scores to evaluate the conversational abilities.

Dataset Preparation We use a dataset comprising 94.1K ShareGPT entries (HuggingFace, 2023b; shareGPT, 2023), which capture user interactions with ChatGPT. Following FastChat (Chiang et al., 2023), we convert HTML content to Markdown, remove non-English conversations, and split long dialogues into segments of up to 2048 tokens. For a fair comparison, all baseline methods are replicated using the same dataset as described above.

Training Details The PEFT method LLaMA-2-LoRA follows the settings: the rank is 8, the alpha is 16, the dropout is 0.05, the learning rate is 2e-5, and the total number of epochs is 3. For full-parameter fine-tuning, all methods follow the same settings: the global batch size is 128, the learning rate is 2e-5, and the total number of epochs is 3. In QFT, we apply channel-wise quantization for all quantizers of model states. The threshold T in the dense-and-sparse quantizer is obtained from 1% of the distribution range (see Appendix D for details).

B The Standard Lion Procedure

Here, we show the standard Lion procedure with full-precision calculations in Algorithm 3.

Algorithm 3 Lion Optimizer

given $\beta_1, \beta_2, \lambda, \eta, f$
initialize $\theta_0, m_0 \leftarrow 0$
for $t = 1, 2, \cdots, T$ **do**
  $g_t \leftarrow \nabla_\theta f(\theta_{t-1})$
  update model parameters
  $c_t \leftarrow \beta_1 m_{t-1} + (1-\beta_1) g_t$
  $\theta_t \leftarrow \theta_{t-1} - \eta(\text{sign}(c_t) + \lambda\theta_{t-1})$
  update EMA of g_t
  $m_t \leftarrow \beta_2 m_{t-1} + (1-\beta_2) g_t$
end for
return θ_t

C Analysis of Values of $\frac{|\Delta|}{\sigma_\delta}$

Lemma 1 shows when $|\Delta| \geq 1.645\sqrt{\beta_1^2\delta_m^2 + (1-\beta_1)^2\delta_g^2}$, there is a 95% probability that $\text{sign}(\Delta)$ is invariant to quantization. Here, we experimentally verify that the condition hold well. Specifically, we sample $\frac{|\Delta|}{\sigma_\delta}$ in different iterations of different layers. σ_δ is obtained by maximum likelihood estimation based on the observed value. The results are presented in Table 4. As we can see, at least 97.9% of cases are greater than 1.645, satisfying the above condition. We also randomly select 1000 samples and visualize their values, as shown in Figure 7.

D Discussion on Outlier Thresholds of Weight Quantizer

In this section, we discuss the selection and updating strategies for outlier thresholds in dense-and-sparse quantizers. We first report the memory and accuracy of dense-and-sparse quantizers using different percentage thresholds, and the results are shown in Table 5. The accuracy, i.e., the degree of distributional approximation of the quantizers, is evaluated by L_2 distance between de-quantized

Table 4: Percentage distribution of $\frac{|\Delta|}{\sigma_\delta}$ in different iterations of different layers. We can see that more than 97.9% of cases are greater than 1.645.

Layer	Iteration	$\frac{\lvert\Delta\rvert}{\sigma_\delta} < 1.645$	$\frac{\lvert\Delta\rvert}{\sigma_\delta} \geq 1.645$
0	200	1.6%	98.4%
	1200	1.3%	98.7%
15	200	1.1%	98.8%
	1200	1.1%	98.9%
31	200	2.1%	97.9%
	1200	1.9%	98.1%

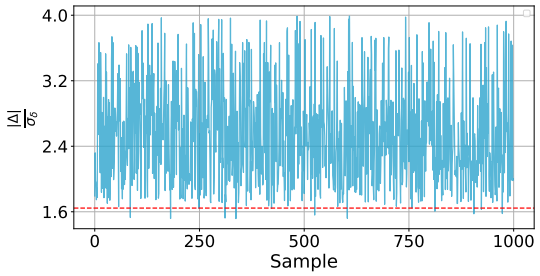


Figure 7: Display of 1000 samples sampled at the 200th training step of the final down projection layer.

weights $\hat{\boldsymbol{W}}$ and full-precision weights $\boldsymbol{W}$, where the quantized weights are from the final down projection layer.

Table 5: Comparison of memory (in GB) and accuracy of dense-and-sparse quantizers using different percentage thresholds for weights. Here, accuracy is measured by L_2 distance between dequantized $\hat{\boldsymbol{W}}$ and full-precision $\boldsymbol{W}$.

Percentile	0	0.45%	1.0%	3.0%	5.0%
Memory	7.06	7.23	7.42	8.23	9.16
L_2 Distance	436	0.846	0.619	0.566	0.479

The benefits of employing matrix decomposition in dense-and-sparse quantizers are readily evident from the results. When the percentile is set to 0, the quantizer no longer filters out the outliers and degenerates into a standard uniform quantizer, resulting in intolerable quantization errors. A noteworthy value suggested in SqueezeLLM (Kim et al., 2023) is 0.45%. However, it's essential to acknowledge that this tight bound is primarily designed for inference scenarios. To accommodate potential fluctuations during training, some level of relaxation in the thresholds is necessary. To this end, we assess the relaxed constraints at percentiles of 1.0%, 3.0%, and 5.0%. The findings reveal that, although the 3% and 5% percentiles yield a slight boost in accuracy, they also incur higher memory usage. Consequently, we advocate for the 1% percentile as a more favorable choice for achieving a balance between memory conservation and accuracy.

We also explore the update frequency of the thresholds. In order to maintain a high training throughput, we adopt a strategy of lazy updates, meaning that the thresholds set in the first iteration are reused in subsequent iterations. This approach is viable due to the smooth numerical updates of the weights during the fine-tuning process across neighboring iterations. Additionally, to our surprise, we discover that the initial thresholds remain effective for an entire epoch. As a result, we only update the thresholds once at the beginning of each epoch.

E QUALITATIVE ANALYSIS OF CONVERSATIONAL ABILITIES

In the experimental section, we demonstrate the effectiveness of QFT through quantitative evaluations of LLMs' performance on a variety of benchmarks. Despite the large number of benchmarks, the results of the machine evaluations are not considered to be comprehensive, and in some cases, their reliability may be subject to scrutiny, e.g., the "shortcut" issue. Consequently, in the following, we intuitively show and compare examples generated by LLMs, aiming to further verify the conversational abilities through qualitative analysis.

[Writing Ability]

Question: Write a captivating movie review for a recently released science fiction film, discussing its plot, characters, and special effects.

LLaMA-2-7B-FT-Adam: "The Last Voyage" is a thrilling science fiction film that follows a group of astronauts on a mission to explore a distant planet. When they arrive, they discover that the planet is inhabited by a mysterious alien race. As they try to communicate with the aliens and gather data, they realize that their mission may be more dangerous than they ever imagined. With stunning special effects and a gripping plot, "The Last Voyage" is a must-see for science fiction fans.

LLaMA-2-7B-QFT: "Intergalactic Intrigue" is a thrilling science fiction film that takes viewers on a journey through the stars. The film follows a group of space explorers as they embark on a mission to uncover the secrets of an ancient alien civilization. The film's special effects are top-notch, with stunning visuals that transport viewers to a world of wonder and discovery. The characters are well-developed and likable, with each member of the crew bringing their own unique skills and personalities to the table.

[Roleplay Ability]

Question: As a superhero, how would you explain your origin story to a curious child?

LLaMA-2-7B-FT-Adam: I was just like any other kid, until one day I was exposed to a strange chemical that gave me incredible powers. I learned to control the elements and use them to protect the innocent. I became known as the Elemental Guardian, and I've been fighting crime and saving the day ever since.

LLaMA-2-7B-QFT: I was just like any other kid, playing in the park and having fun. But one day, I was hit by a strange ray that gave me superpowers. I could fly, turn invisible, and even control the weather! I decided to use my powers to help people and fight crime. That's how I became the superhero you know and love!

[Common-sense Ability]

Question: In a world where automation is becoming increasingly prevalent, is it more important to prioritize job creation or technological progress?

LLaMA-2-7B-FT-Adam: Job creation is essential for a thriving economy, as it provides individuals with employment opportunities and a source of income. While technological progress is important for driving innovation and improving efficiency, it is crucial to balance technological advancements with the creation of new jobs. This can be achieved by investing in education and training programs that help workers develop the skills necessary to adapt to new technologies and by promoting policies that support small businesses and entrepreneurship. By prioritizing both job creation and technological progress, we can ensure a more stable and prosperous future for all.

LLaMA-2-7B-QFT: In the short term, job creation is a critical concern for policymakers and businesses. The unemployment rate remains high, and many people are struggling to find work. However, in the long term, technological progress is essential for driving economic growth and improving the standard of living for everyone. Automation has the potential to increase productivity and efficiency, which can lead to higher wages and better working conditions for employees. However, it can also lead to job displacement, particularly in industries where automation is most prevalent.

[Fermi Ability]

Question: How many snowflakes fall during a typical winter? Try to explain your answer. Your explanation should take the reader through your reasoning step-by-step.

LLaMA-2-7B-FT-Adam: It is difficult to determine the average number of snowflakes that fall during a typical winter, as it can vary greatly depending on location, climate, and other factors. However, it is estimated that on average, a single snowflake can have up to 100,000 ice crystals per flake. Over the course of a typical winter, this could result in billions of snowflakes falling from the sky.

LLaMA-2-7B-QFT: To answer this question, we need to consider the average snowfall for a typical winter in a given location. In the United States, for example, snowfall tends to be heaviest in the northern states, where the average snowfall for a typical winter is around 60 inches. In the southern states, snowfall is much less common and the average snowfall for a typical winter is around 10 inches.

[Counterfactual Ability]

Question: What if the Internet had been invented during the Renaissance period?

LLaMA-2-7B-FT-Adam: The Renaissance would have been a much different time period if the internet had been invented. The flow of information would have been much faster and more efficient, leading to a greater exchange of ideas and innovations. The world would have been connected in ways never before seen, allowing for the rapid spread of new ideas and technologies.

LLaMA-2-7B-QFT: The Renaissance was characterized by a renewed interest in classical learning and the arts, and the Internet would have provided a way for people to access and share this knowledge. It would have also facilitated the exchange of ideas and the development of new technologies, leading to even greater advancements in the fields of science, medicine, and engineering.

[Generic Ability]

Question: Can you explain the basics of quantum computing?

LLaMA-2-7B-FT-Adam: Quantum computing is a type of computing that uses quantum bits (qubits) to perform calculations, unlike traditional computers that use binary.

LLaMA-2-7B-QFT: Quantum computing is a type of computing that uses quantum-mechanical phenomena, such as superposition and entanglement, to perform operations on data. These phenomena allow quantum computers to perform certain calculations much faster than classical computers.

[Knowledge Ability]

Question: Explain the process of natural selection and how it contributes to the evolution and adaptation of species.

LLaMA-2-7B-FT-Adam: Natural selection is the process by which species adapt to their environment through the survival and reproduction of individuals with favorable traits. Over time, these favorable traits become more common in the population, leading to the evolution of new species.

LLaMA-2-7B-QFT: Natural selection is a process where species that are better adapted to their environment tend to survive and reproduce more than those that are less adapted. This leads to the survival of the fittest and the gradual evolution of species over time.