What's the main reason for not submitting this paper at this moment?

" I think that there are still some major issues in the theory & algorithm implementations, and these issues cannot be fixed before the deadline (actually we may need another 1 or 2 months to fix these issues)."

QuZO: Quantized Zeroth-Order Fine-Tuning for Large Language Models on Low-Precision Platforms

Jiajun Zhou[1,3†], **Ziyue Liu**[1], **Yifan Yang**[1], **Yequan Zhao**[1], **Kai Zhen**[2], **Ngai Wong**[3]

Athanasios Mouchtaris[2], **Siegfried Kunzmann**[2], **Zheng Zhang**[1]

[1]University of California, Santa Barbara, CA

[2]Amazon Alexa AI Amazon Artificial General Intelligence (AGI)

[3]The University of Hong Kong

Abstract

Large language models (LLM) are often quantized to a low precision in order to improve inference speed and to reduce hardware cost (e.g., memory and energy) when deployed on resource-constraint computing platforms. Can we leverage a limited-precision inference engine to perform fine-tuning with minimal hardware adjustment? To achieve this goal, this paper presents a quantized zeroth-order method, QuZO, for fine tuning LLMs on a low-precision hardware platform. QuZO is tailored for different precisions with integer (INT) and floating-point (FP) arithmetic to achieve high training accuracy in low-precision settings. Furthermore, the memory-efficient QuZO even achieves better fine-tuning accuracy than first-order training when the precision is very low (e.g., below INT8), because QuZO does not use the error-prone straight-through estimator that is widely used in truly quantized first-order training. Our method demonstrates competitive accuracy across various downstream tasks. Our result shows that QuZO can reduce the memory consumption by $5\times$ in LLaMa-7B fine-tuning compared to first-order optimization with the FSDP engine when using an INT8 quantization scheme.

1 Introduction

There are several aspects to consider when quantizing large language models (LLMs) Brown et al. [2020], Yuan et al. [2024], including Fully Quantized Training (FQT) Xi et al. [2023], Quantization-Aware Training (QAT) Liu et al. [2023b], and Post-Training Quantization (PTQ). In traditional research, FQT methods offer quantizers and dequantizers in full-precision computational graphs and utilize low-precision CUDA kernels for implementation. Various low-precision training methods have been proposed to accelerate training while maintaining accuracy. FP16 typically reduces memory usage by approximately half, speeds up training, and maintains accuracy on most tasks. For example, the LLaMa-2 model is commonly stored in BF16 format, requiring fewer GPUs for loading and opening up opportunities for fine-tuning. Additionally, FP8 training Wang et al. [2018], implemented by NVIDIA and integrated into the Transform Engine on the H100 GPU, is gaining traction.

Furthermore, it is widely recognized that INT data types are more cost-effective than FP formats. Recent advancements have reduced numerical precision to 4-bit, with successful implementations of INT4 weight/activation and FP4 gradients. The introduction of a 4-bit logarithmic numerical format has further improved model accuracy. Another approach Xi et al. [2023] involves the use of a Hadamard quantizer and INT4 Cutlass GEMM kernel to accelerate training. However, FQT methods require the design of forward and backward passes with a quantization strategy, with gradient

What about FP8? Or NF4?

†Work undertaken during the visit at UC Santa Barbara

Preprint. Under review.

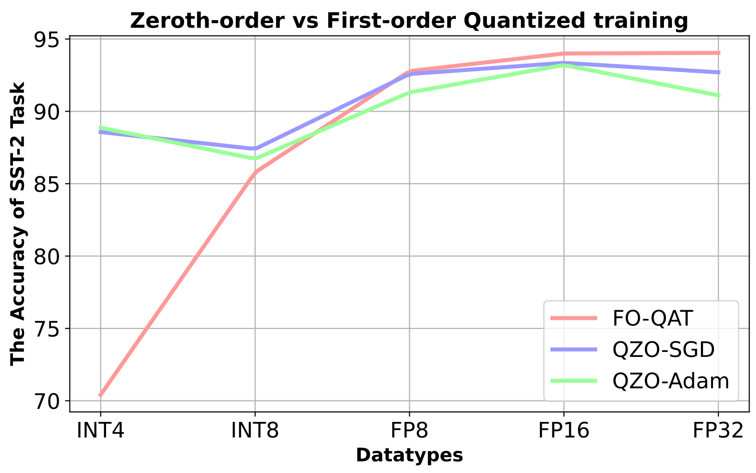


Figure 1: Comparison of the SST-2 task across QuZO and first-order (FO) methods, and various precision levels of INT/FP arithmetic in the RoBERTa-Large model.

quantization being a critical aspect, often relying on the straight-through estimator, which introduces significant overhead during the training process.

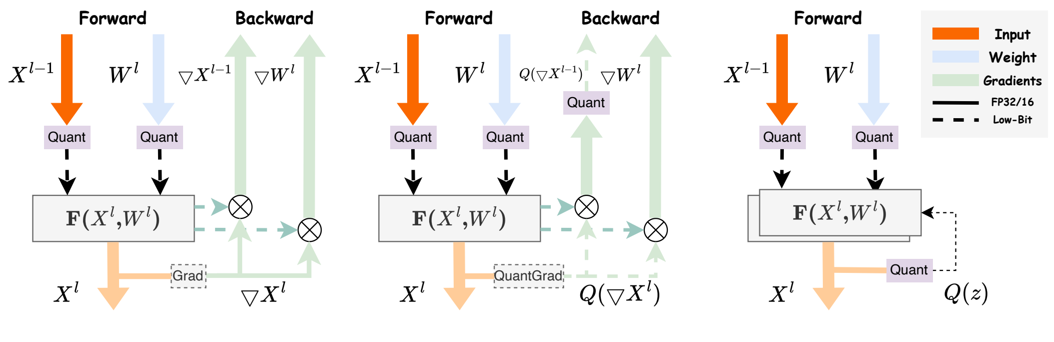
From a different perspective, the zeroth-order method offers a backpropagation-free training flow and further reduces overhead during LLM fine-tuning. MeZO Malladi et al. [2024], a recently proposed zeroth-order method for fine-tuning LLMs, has demonstrated impressive performance. However, full-precision ZO fine-tuning still faces challenges in deploying LLMs on resource-constrained hardware, such as Edge FPGAs or embedded systems.
To overcome these bottlenecks, we proposed a novel quantized zeroth-order method to solve the issues of the high computational requirement of LLMs. Our contributions are :

- We demonstrate the effectiveness of the Quantized ZO method by showcasing INT4/8 and FP8 for LLM fine-tuning. Our QuZO approach enjoys better performance than the traditional QAT and FQT methods by using first-order optimization across various scales of models and tasks.
- We introduce a 4-bit perturbation QuZO method to update the quantized model during training, enhancing the efficiency and effectiveness of the training process. Besides, We conduct a comprehensive analysis of quantization loss on the QuZO method, providing insights into the comparative performance and trade-offs associated with each approach.
- We propose an adaptive approach for selecting between INT and FP during quantization, aiming to further accelerate training with minimal accuracy loss. Additionally, an outlier detector is introduced to distinguish normal and outlier values, assigning adaptive FP quantization for outliers.

2 Related Work

Zero-order method Zero-order (ZO) optimization techniques do not require derivatives during training. It significantly reduces memory consumption from a first-order (FO) gradient-based counterpart method. MeZO Malladi et al. [2024] enables the scalability of ZO optimization to large-scale LLMs while maintaining memory efficiency. There are variable ZO methods including ZO-SGD Ghadimi and Lan [2013] employed for LLM fine-tuning, ZO-Sign-SGD Liu et al. [2018] using signed-based gradient estimation, and the ZO-Adam Chen et al. [2019] optimizer. Sparse MeZO seeks a sparse perturbation to do LLM fine-tuning. In this paper, we further push the ZO fine-tuning into low-precision including quantized model and gradient. It unlocks the possibility of LLM fine-tuning on the resource-constrained hardware.

Quantization of LLMs Model quantization is commonly used for hardware deployment and results in a compact model. LLM.int8() Dettmers et al. [2022] only reduces the precision of model weights



(a) Quantization-Aware Training

(b) Full Quantized Training

(c) Quantized Zeroth-order Training

Is FP32/16 used? The output is missing in the legend.

Figure 2: Computational graphs for quantized first-order training and QuZO settings.

and keeps the outlier in FP16. SmoothQuant Xiao et al. [2023] proposed a fine-grained quantization strategy with only INT8 operations. QLLM Liu et al. [2023a] introduces an adaptive channel reassembly technique to solve the outlier problem. Apart from these, another way is to do quantization-aware training like LLM-QAT Liu et al. [2023b] work which proposed a data-free strategy to reduce quantization-level to 4bit. Besides, 4-bit transformer training Xi et al. [2023] proposed Hadamard quantization and sparse gradients to speed up training. Unlike QAT or FQT, we proposed the quantized zeroth-order(QZO) training method to avoid backward computation which requires only an inference framework to fine-tune models.

3 Method

Neural network training is typically an iterative optimization process involving stochastic gradients during forward and backward passes. Traditional frameworks for fully quantized training (FQT) often leverage low-precision integer or floating-point arithmetic to accelerate both forward and backward propagation. However, we present a quantized forward-only approach that eliminates the need for derivatives in optimization, significantly reducing overhead and simplifying the fine-tuning of large language models (LLMs). In this study, we aim to expedite the training of LLMs by employing quantized models and ZO methods. Our approach begins with the introduction of a Quantized Zeroth-Order (QuZO) framework. Here, forward propagation is conceptualized as a series of linear and non-linear operations (such as GeLU, softmax, etc.). During our quantized training process, we initialize the model in a quantized state and leverage QuZO techniques to update parameters during training. Additionally, we leverage the Cutlass INT4 or INT8 kernel to further enhance training acceleration, as linear operations in LLMs can often be interpreted as matrix multiplications. For LLMs, we can apply quantization on both activation and weight matrices (weight-activation quantization) or just the latter (weight-only quantization). Figure 2 presents an overview of our quantized framework tailored for LLMs.

3.1 Integer Quantization Function

Building on insights from the study by XX, which highlighted the prevalence of outlier issues in language models(LMs), our research focuses on addressing these challenges. Specifically, we commence by assessing the efficacy of uniform quantization techniques for LLMs during zero-order (ZO) training. The quantization and dequantization function for a quantized parameter $\overline{x}$ can be explained as the following equation. (1):

$$\overline{x} = s \cdot \text{Dequant}(\lceil \text{Clamp}(\text{Quant}(\frac{x-\beta}{s}), \min, \max) \rfloor + \beta) \tag{1}$$

Algorithm 1 Quantized Zeroth-Order Training

```
Require: LLM model parameters $x \in \mathbb{R}^d$, learning rate $n_t$, $T$ is the step, perturbation scaling factor $\epsilon$.
 1: Initial Pre-trained Model to Quantized Model
 2: $\overline{x} = Quant(x)$                          ◁ Quantize FP32 model to 4/8-bit model including INT and FP.
 3: Random seed $s$ and QuZO fine-tuning initiated.
 4: for t in $T$ do
 5:     $\overline{x} \leftarrow \text{QuantizedPerturb}(x, \epsilon, s)$     ◁ 2/4/8-bit INT Perturbation used to update $x$.
 6:     $l_+ \leftarrow \text{ZOForward}(\overline{x}, B)$     ◁ ZO forward pass with inference-only framework.
 7:     $\overline{x} \leftarrow \text{QuantizedPerturb}(x, -2\epsilon, s)$
 8:     $l_- \leftarrow \text{ZOForward}(\overline{x}, B)$
 9:     $\overline{x} \leftarrow \text{QuantizedPerturb}(x, \epsilon, s)$
10:     $\text{projected}_{\text{grad}} \leftarrow (l_+ - l_-)/2\epsilon$
11:     for $x_i$ in $x$ do
12:         $z_i \sim \text{Quant}[\mathcal{N}(0,1)]$     ◁ INT-only perturbation generated using normal distribution.
13:         $x_i \leftarrow x_i + \epsilon z_i$     ◁ Update the model parameter.
14:     end for
15: end for
16: return $\overline{x}$     ◁ Return the quantized-version model.
```

Let's say in IN4, w is updated to w' which would still be quantized to w.
How does Quzo handle this?

where s is the scaling factor for the quantization function, β is the zero-point value, min, and max are the minimum and maximum values for the $Clamp()$ function. $Quant$ and $Dequant$ stand for the quantization and dequantization functions. There are two quantization categories, one is for symmetric quantization, $s = \frac{max(|X|)}{2^N-1}$, and β = 0. Another is for asymmetric quantization, $s = \frac{max(|X|)-min(|X|)}{2^N-1}$ and $\beta = min(|X|)$. In this way, the scale s and zero-point value β can be learned statistically in the learned step size quantizer (LSQ), SmoothQuant and Q-BERT work Shen et al. [2020].

3.2 Floating Point Numbers

The floating point number representation scheme contains a mandatory sign, multiple exponent bits, and mantissa bits. Specifically, a set of FP numbers in R representation is defined in Eqn. (2).

$$\mathbf{f}(\mathbf{x}) = \begin{cases} 0, & 0 \\ 2^n, & max \\ (-1)^s \times 2^{i-b} \times \left(1 + \frac{x}{2^k}\right), & others \end{cases} \tag{2}$$

where n refers to the total number of bits, i stands for the length of the exponent bits, k is the length of the power-of-2 scaling mantissa bits with shared bias value b of encoded variable length range bits, and x represents the decimal number of the fraction field. Certainly, The dynamic range of floating-point formats is determined by the exponent's width, which scales the mantissa. This non-uniform spacing allows for greater accuracy with smaller values, making floating-point formats well-suited for deep learning workloads, including LLMs, which often exhibit long-tailed normal distributions.

3.3 Quantized Zeroth-Order Optimization

Zeroth-order(ZO) optimization servers as a backpropagation-free alternative to the first-order method. Almost all works were built on FO optimization when applying QAT or FQT techniques for model quantization. FO gradients would add much more complicated computational overhead and should be designed carefully when doing FQT like this work. Thus, the ZO optimizer typically replaces the FO gradient with the ZO gradient estimate as the descent direction. Fine-rune pre-trained LLMs are particularly intriguing by using ZO optimization. This offers the scalability of ZO methods to LLMs with low memory overhead. MeZO introduced a memory-efficient ZO stochastic gradient descent (ZO-SGD) algorithm to efficiently fine-tune LLMs with above 60 billion parameters with some PEFT approaches like LoRA. For instance, we extend the ZO-SGD optimizer for quantized LLM fine-tuning. In the following, we first introduce a quantized ZO gradient estimator. Randomized Gradient Estimators (RGE) have been widely used to fine-tune LLMs.

Table 1: Comparison of Memory Storage for Different Training Methods on A100 GPUs.

Hardware Cost	Method	Training Memory(GB)
1 x A100	QuZO	29.6
1 x A100	MeZO	39.33
4 x A100	FO(FSDP)	4 x 37.5

Definition 1 (Quantized Randomized Gradient Estimator). Given a model with parameters $x \in \mathbb{R}^d$ and a loss function L, the original RGE is expressed using SPSA and estimates the gradients on a minibatch dataset B. A minibatch $B \in$ a dataset D.

$$\widehat{\bigtriangledown} L(x, B) = \frac{1}{q} \sum_{i=1}^{q} \frac{L(x+\overline{u_i}\epsilon, B) - L(x-\overline{u_i}\epsilon, B)}{2\epsilon} \overline{u_i} \text{ (RGE)}$$

where $\overline{u_i}$ is a low-precision random direction vector with $u_i \sim N(0,1)$, q is the number of function queries and ϵ is the perturbation scale factor. We applied RGE by setting $q = 1$ and $u_i = u$. The rationale behind RGE requires only two forward passes through the quantized model to compute the gradient estimate. As $\epsilon \to 0$, the RGE $\widehat{\bigtriangledown} L(x)$ can be equal to $L'(x, u)u$. Therefore, the $\widehat{\bigtriangledown} L(x)$ can be interpreted as an approximation of the backpropagation in any optimizer. In Algorithm 1, we provide the in-place implementation with only inference hardware support. That is a crucial benefit for the hardware-efficient LLM's fine-tuning.

3.3.1 Quantized Perturbation

Prior works like Malladi et al. [2024], Liu et al. [2023a] use FP32 to update the gradient during training. We further reduce the random perturbation u to one memory-efficient arithmetic - Integer(2/4/8-bit). The idea is quite simple, u typically represented with higher precision (e.g., FP32), is converted into 2/4/8-bit INT. This conversion involves determining a scaling factor to map the range of FP32 to the desired range of integers. Each FP is then multiplied by this scaling factor and rounded to the nearest integer, resulting in quantized integer numbers. Quantized values are clipped to fit within the representable range. During QuZO fine-tuning, the quantized integers can be converted back to floating-point numbers using the reverse scaling factor. While quantization effectively reduces memory usage and computational complexity, it may introduce quantization errors that can impact model accuracy.

Storage Efficiency of QuZO. To demonstrate the hardware efficiency of QuZO, we employ the Cutlass INT8 Kernel to showcase memory efficiency. The torch.int library, widely used in PTQ frameworks such as SmoothQuant Xiao et al. [2023] and AWQ Lin et al. [2023], serves as the basis for QuZO. Therefore, QuZO simply utilizes a developed INT linear kernel in the CUDA inference engine for LLM fine-tuning purposes. In the LLaMa-2 model, our method reduced memory consumption by 25% for full-parameter tuning using a single A100 GPU compared to the MeZO approach in Table 1. We believe it can be further reduced if we fully apply the INT engine in each linear and non-linear layer. This could be our next step for the QuZO method on the CUDA optimization.

3.3.2 Adaptive low-precision arithmetic selection

Normally, simply assigning the same datatypes for QuZO with 4/8-bit weight/activation could drop the accuracy due to the sensitivity of activation. Taking the quantization error into consideration, our QuZO framework can efficiently calculate the score for INT and FP quantization with the same bit-width and choose the best format for the selected layer. Existing works like DyBit, etc also take this into their quantization strategy. Since the activations have outlier issues and pose a challenge between quantization granularity and representable data range. We use the mean squared error (MSE) as the metric to calculate the quantization loss. We defined this by the following equation. (4):

$$MSE = E[(x - \overline{x})] = \int (x - \hat{x})^2 p(x)\, dx \tag{3}$$

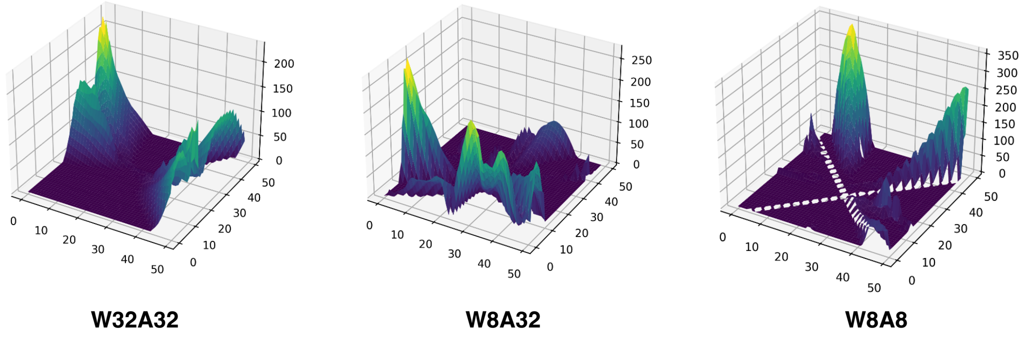


Figure 3: The loss landscape of the RoBERTa-large model under different quantization bits. The notations W and A mean the bits for weights and activation.

$$MSE = \int_{x_{\text{min}}}^{x_{\text{max}}} (Q(w) - w)^2 p(w)\, dw \quad + \int_{x_{\text{max}}}^{\infty} (x_{\text{max}} - w)^2 p(w)\, dw \quad + \int_{-\infty}^{x_{\text{min}}} (w - x_{\text{min}})^2 p(w)\, dw \tag{4}$$

where x and $\overline{x}$ are the FP32 and quantized value, and p(x) stands for the probability density function. The neural network weights are a random variable w $p_w(w)$. The quantization range is defined between x_{min} and x_{max}. In this way, our QuZO framework chooses the most suitable datatype for each tensor and only executes the searching algorithm once before fine-tuning. Many existing works proved that the tensor distribution of one well-trained model stays constant during the fine-tuning stage. Through our adaptive type searching algorithm, we found that integer (INT) quantization is much more suitable for weight quantization and offers better hardware efficiency. Floating-point (FP) quantization is primarily chosen for activation quantization to maintain good accuracy. This adaptive datatype selection offers a more accurate QuZO fine-tuning process on current-generation GPUs since it supports both FP and INT formats.

3.3.3 Outlier Quantization

From the analysis of prior works Liu et al. [2023b], the outlier of the transformers is significant. LLM-FP4 Liu et al. [2023a] quantizations demonstrate a similar observation and achieve acceptable accuracy with FP4 quantization for LLMs due to outliers. Therefore, AWQ keeps a roughly 1% salient channel in FP16 can benefit the transformer models. To accelerate the quantization process without introducing overhead, we proposed an outlier detector that can distinguish the normal and outlier values. The outlier detector can distinctly identify the salient data. Let us take a Signed INT4 quantization as an example, we choose the binary 1000 as an outlier label to decide if the outlier value appears in the selected tensor array, where 1000 represents the value of -8. We sacrificed one binary code to detect the outlier and applied the 4-bit adaptive biased float to quantize the outlier.

3.3.4 Loss Landscape Analysis for QuZO

The effectiveness of ZO LLMs fine-tuning, unlike traditional ZO training, stems from the initiation of the fine-tuning process at points near the optimal region of the loss landscape. This theoretical analysis supports the hypothesis proposed in Lemma 3 Malladi et al. [2024] that the convergence rate of ZO LLM tuning is related to the effective rank of the model parameters' Hessian, rather than to the number of parameters as suggested in traditional ZO literature Ghadimi and Lan [2013], Liu et al. [2020]. Furthermore, the Lemma provides specific proof of global convergence for fine-tuning, revealing the relationship between the steps required for convergence and model-specific factors, such as the effective rank r and the L-Lipschitz smoothness constant L. The lemma demonstrates that the ZO-SGD algorithm can converge after t iterations, where t is given by:

$$t = \mathcal{O}(r, \alpha, \log(\mathcal{L}(\boldsymbol{\theta}_0) - \mathcal{L}^*)), \tag{5}$$

where $\mathcal{L}(\theta)$ represents the loss function, and $\alpha = 2NL^2$ is a constant related to the L-Lipschitz smoothness factor L and the batch size N.

For the quantized case, the effective rank of the Hessian matrix of specific model parameters does not change significantly, as the model architecture remains fixed. Consequently, the L-Lipschitz smoothness constant L becomes the critical factor influencing performance changes under the ZO fine-tuning setup. Thus, observing the smoothness of the model's loss landscape can provide solid evidence for the effectiveness of ZO fine-tuning in quantized LLMs.

To visualize the loss landscape of the models with varying degrees of quantization, we present it in Fig. 3. According to this figure, it is evident that the smoothness of the loss landscape does not significantly change when reducing the quantization bits of weights, particularly near the optimal region. This observation aligns with our experimental results, indicating that ZO quantized fine-tuning performs well under weight quantization scenarios, maintaining accuracy even at INT4 bits. In contrast, quantization of activations significantly affects the loss landscape's smoothness, as evidenced by the large fluctuations observed in the ridge of the third sub-figure in Fig. 3. This fact suggests that activation quantization presents a more challenging but still possible scenario for training with a ZO optimizer, which aligns with the slight accuracy drop for activation quantization training in our experiment. Differently, compared to ZO optimization, FO optimization suffers from the inaccuracies introduced by the straight-through estimator, as previously mentioned, which underperforms in low-bit scenarios.

4 Experiments

Preliminary experiments show that QuZO works for various data types and different precisions. Besides, there is a bunch of work focused on LLM quantization. Past works have demonstrated the PTQ approach is quite an efficient method for low-precision integer inference scenarios. INT4 FQT approach enables 4-bit transformer training but requires special forward and backpropagation which is non-general for different hardware platforms. LLM-QAT still applied the first-order method to fine-tune LLMs. To fairly comparison, we use the first-order fine-tuning as a baseline with the same quantization algorithm, such as symmetric and asymmetric strategies. All fine-tuning with backpropagation experiments uses Adam, though we also report results with ZO-Adam and ZO-SGD optimizer. We adopt both medium-sized models (RoBERTa-Large Liu et al. [2019] and BERT-Large Kim et al. [2021]) and large decode-based LMs (OPT-1.3B and LLaMa-2 7B Touvron et al. [2023]) in few-shot settings. QuZO uses significantly less memory than FO but requires more training steps.

4.1 Main Results

We first conduct experiments with RoBERTa-large on sentiment classification and natural language classification tasks. We follow prior works in low data resource settings which can be sampling k examples per class for k = 16 or 512. QuZO is running for 100K steps and 1000 fine-tuning steps and its step is quite faster than the one first-order fine-tuning step.

Results of RoBERTa-Large We use the pretrained RoBERTa-Large model and evaluate the performance of our method on the SST-2, SNLI, SST-5, RTE, and MNLI datasets. In the RoBERT experiments, QuZO and FO methods use naive quantization to do fine-tuning. From Table 2, QuZO can work consistently for smaller LMs during 4/8-bit fine-tuning. Compared with the first-order fine-tuning, QuZO successfully preserves model accuracy with W4A8 quantization and also performs better in W8A8 quantization by adopting the INT format. We further compare the results with recent LLM-QAT work, LLM-QAT applied a dynamic quantization with an asymmetric strategy which no doubt improved performance a lot. But in W4A8 quantization, QuZO can be better than the LLM-QAT approach in these 5 tasks.

Results of LLaMA models LLaMa models are currently widely used open language models with superior performance on many tasks. Through our experiments, our methods can handle the difficulty of quantization of very large-scaled LLMs whose activation is difficult to quantize. We extend the QuZO to the LLaMa-2 model and select SuperGLUE tasks and generation tasks. Recent LLM quantization works like LLM.int8() and SmoothQuant mainly focus on 8-bit quantization. We compare QuZO with SmoothQuant in the W8A8 quantization setting in Table 3, and QuZO achieves comparable performance on multiple-choice and generation tasks. Meanwhile, QuZO surpasses the FO method for W4A8 and W8A8 quantization in all 11 tasks.

QuZO w. o. RGE: is QRGE the accuracy bottleneck here?

Table 2: Experimental Results on RoBERTa-large (350M parameters) with Prompts. QuZO, leveraging full-parameter tuning, outperforms FO and LLM-QAT by a significant margin. Additionally, it approaches Fine-Tuning (FT) performance while requiring significantly less memory.

Method	#Bits (W-A)	Dtype	SST-2	SNLI	SST-5	RTE	MNLI
FO(Baseline)	8-8	FP	92.78	87.04	55.82	83.98	82.68
QuZO(SGD)	8-8	FP	91.31	81.05	52.45	79.39	73.44
QuZO(Adam)	8-8	FP	91.11	71.03			
FO(Baseline)	8-8	INT	85.78	63.09	39.18	73.93	66.41
LLM-QAT	8-8	INT	93.34	86.72	55.27	84.11	81.58
QuZO(SGD)	8-8	INT	91.02	75.78	51.05	74.51	67.45
FO(Baseline)	4-8	INT	70.4	44.2	29.88	63.38	55.99
LLM-QAT	4-8	INT	77.64	44.47	20.9	50	36.59
QuZO(SGD)	4-8	INT	88.57	53.25	46.84	63.18	53.13

Table 3: QuZO demonstrates superior performance across various tasks in LLaMa-2 7B experiments.

LLaMa-2 7B Model			Classification							Multiple-Choise		Generation	
Method	Datatype	Precision	SST-2	RTE	CB	BoolQ	WSC	WIC	MultiRC	COPA	ReCoRD	SQuAD	DROP
QuZO(Ours)	FP	W8A8		51.98			57.05			81		80.93	
FO (Baseline)	INT	W8A8	86.35	47.29	57.14	61.6	61.53	54.23	50.6	62	33.9	0.03	0.062
SmoothQuant	INT	W8A8	91.05	66.78	64.28	68.7	59.51	66.3	61.5	72	79.1	53.05	29.94
QuZO(Ours)	INT	W8A8		60.64			63.46	52.82		81		77.71	
FO (Baseline)	INT	W4A8	50.8	52.34	33.92	62.4	60.57	51.41		58	13	0.001	0.045
QuZO(Ours)	INT	W4A8								83		78.12	
QuZO(Ours)	INT	W8A32		61.37				57.05		81		80.93	

Results of OPT models OPT models generally have more severe activation issues compared to LLaMA and BLOOM models. It induces more challenges for LLM quantization. Nonetheless, QuZO still works quite well for OPT models and we provide some results of W8A8 quantization in the table 4. It clearly show that our method performs better than the QLLM approach in most tasks.

4.2 4-bit Quantized Perturbation

We next explore the ZO gradient quantization, which can accelerate model training without compromising convergence. Current works only focus on sparse parameter perturbations for reducing gradient estimation variance in RGE. It introduces the masks and applies them to weight perturbations per step. However, we now consider on hardware-efficient side and use low-precision weight perturbation to do ZO gradient estimation in LLM fine-tuning. It becomes evident that choosing 4-bit can maintain training performance and reduce memory overhead. Using a fully quantized I-BERT Kim et al. [2021] as an example, we assign 2/4/8-bit perturbation to update the INT8 model in Table 5. The accuracy dropped within 1% but reduced the around 4-16$\times$ memory storages for the random perturbation parameters. That is a huge benefit for ZO training since the perturbations are generated and calculated many times for one training step. As for other tasks (such as SST2,MR and CR), we adopted 4-bit perturbation to fine-tune the RoBERTa-Large model and the performance remained well compared with full-precision fine-tuning. This technique unlocks various hardware platforms for fine-tuning LLMs without much hardware configuration.

Table 4: Experiments conducted on OPT-1.3B model.

OPT-1.3B Model			Classification							Multiple-Choise		Generation	
Method	Datatype	Precision	SST-2	RTE	CB	BoolQ	WSC	WIC	MultiRC	COPA	ReCoRD	SQuAD	DROP
QLLM	INT	W8A8	82.45	55.59	66.07	63	63.46	52.35	56.8	71	59.9	61.49	OOM
QuZO(Ours)	FP	W8A8											
QuZO(Ours)	INT	W8A8											

Table 5: Low-precision perturbation performance across 2 models and 3 tasks.

Model	Model Precision	Perturbation (#bit)	Task	Performance
I-BERT	INT8	2	SST-2	91.89
I-BERT	INT8	4	SST-2	92.48
I-BERT	INT8	8	SST-2	92.77
RoBERTa-Large	FP32	8	SST-2	92.38
RoBERTa-Large	INT8	4	SST-2	91.51
RoBERTa-Large	INT8	4	MR	87.21
RoBERTa-Large	INT8	4	CR	94.53

4.3 Mixed Datatype QuZO Training

We evaluate the accuracy of QuZO for LLMs under low data resource settings. The inference of LLMs is challenging as it requires significant memory, which still makes their retraining even more resource-consuming. Thus, the QuZO method with BP-free training is more desirable than the QAT and FQT method for LLMs.

4.4 Computational and Memory Efficiency

Finally, we demonstrate the potential of our QuZO method to accelerate neural network training by evaluating our prototypical implementation. We emphasize that our implementation is not fully optimized. For example, the forward computation requires an INT8 MM in the form of X = AB with transpose, while Cutlass only supports Y = AB^T. We also do not fuse the linear operators with nonlinearities and normalizations. Therefore, the results cannot fully reflect the potential of INT training algorithms. A fully optimized implementation requires heavy engineering, which exceeds the scope of our paper.

Operator Speed: We compare the throughput of our method by using developed CUDA kernels, and their average throughput (INT8) with a baseline tensor-core FP16 GEMM implementation (FP16) provided by cutlass on an Nvidia RTX 3090 GPU which has a peak throughput at 142 FP16 TFLOPs and 900 INT8 TFLOPs. As the matrix size grows, the overhead of quantization diminishes and our INT8 operators can be up to 1.6 $\times$ faster compared with FP16 MM. We further analyze the quantization overhead for each operator.

5 Conclusion

We propose a memory-efficient Quantized ZO (QuZO) training method for LLMs. Experiments demonstrate that QuZO effectively optimizes models across different precision levels, outperforming first-order methods in scenarios where the latter struggles. In critical tasks like generation, our approach achieves comparable or superior results to existing first-order methods, particularly in LLaMa-2 models. Importantly, QuZO opens avenues for fine-tuning LLMs on low-bit inference frameworks, paving the way for edge computing applications in the future. We leave it as a future direction.

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.

Xiangyi Chen, Sijia Liu, Kaidi Xu, Xingguo Li, Xue Lin, Mingyi Hong, and David Cox. Zo-adamm: Zeroth-order adaptive momentum method for black-box optimization. *Advances in neural information processing systems*, 32, 2019.

Tim Dettmers, Mike Lewis, Younes Belkada, and Luke Zettlemoyer. Gpt3. int8 (): 8-bit matrix multiplication for transformers at scale. *Advances in Neural Information Processing Systems*, 35: 30318–30332, 2022.

Saeed Ghadimi and Guanghui Lan. Stochastic first-and zeroth-order methods for nonconvex stochastic programming. *SIAM journal on optimization*, 23(4):2341–2368, 2013.

Sehoon Kim, Amir Gholami, Zhewei Yao, Michael W Mahoney, and Kurt Keutzer. I-bert: Integer-only bert quantization. In *International conference on machine learning*, pages 5506–5518. PMLR, 2021.

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.

Jing Liu, Ruihao Gong, Xiuying Wei, Zhiwei Dong, Jianfei Cai, and Bohan Zhuang. Qllm: Accurate and efficient low-bitwidth quantization for large language models. *arXiv preprint arXiv:2310.08041*, 2023a.

Sijia Liu, Pin-Yu Chen, Xiangyi Chen, and Mingyi Hong. signsgd via zeroth-order oracle. In *International Conference on Learning Representations*, 2018.

Sijia Liu, Pin-Yu Chen, Bhavya Kailkhura, Gaoyuan Zhang, Alfred O Hero III, and Pramod K Varshney. A primer on zeroth-order optimization in signal processing and machine learning: Principals, recent advances, and applications. *IEEE Signal Processing Magazine*, 37(5):43–54, 2020.

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

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*, 2023b.

Sadhika Malladi, Tianyu Gao, Eshaan Nichani, Alex Damian, Jason D Lee, Danqi Chen, and Sanjeev Arora. Fine-tuning language models with just forward passes. *Advances in Neural Information Processing Systems*, 36, 2024.

Sheng Shen, Zhen Dong, Jiayu Ye, Linjian Ma, Zhewei Yao, Amir Gholami, Michael W Mahoney, and Kurt Keutzer. Q-bert: Hessian based ultra low precision quantization of bert. In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 34, pages 8815–8821, 2020.

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*, 2023.

Naigang Wang, Jungwook Choi, Daniel Brand, Chia-Yu Chen, and Kailash Gopalakrishnan. Training deep neural networks with 8-bit floating point numbers. *Advances in neural information processing systems*, 31, 2018.

Haocheng Xi, Changhao Li, Jianfei Chen, and Jun Zhu. Training transformers with 4-bit integers. *Advances in Neural Information Processing Systems*, 36:49146–49168, 2023.

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

Zhihang Yuan, Yuzhang Shang, Yang Zhou, Zhen Dong, Chenhao Xue, Bingzhe Wu, Zhikai Li, Qingyi Gu, Yong Jae Lee, Yan Yan, et al. Llm inference unveiled: Survey and roofline model insights. *arXiv preprint arXiv:2402.16363*, 2024.