MUZO: Leveraging Multiple Queries and Momentum for Zeroth-Order Fine-Tuning of Large Language Models

Yuezhang Peng[1,2], Yuxin Liu[2], Fei Wen[3]*, Xie Chen[1]*
[1]X-LANCE Lab, School of Computer Science, MoE Key Lab of Artificial Intelligence, Shanghai Jiao Tong University [2]SJTU Paris Elite Institute of Technology, Shanghai Jiao Tong University [3]School of Information Science and Electronic Engineering/School of Integrated Circuits, Shanghai Jiao Tong University
{pengyuezhang, emotion-xin, wenfei, chenxie95}@sjtu.edu.cn

Abstract

Fine-tuning pre-trained large language models (LLMs) on downstream tasks has achieved significant success across various domains. However, as model sizes grow, traditional first-order fine-tuning algorithms incur substantial memory overhead due to the need for activation storage for back-propagation (BP). The BP-free Memory-Efficient Zeroth-Order Optimization (MeZO) method estimates gradients through finite differences, avoiding the storage of activation values, and has been demonstrated as a viable approach for fine-tuning large language models. This work proposes the **MU**ltiple-query Memory Efficient **Z**eroth-**O**rder (**MUZO**) method, which is based on variance-reduced multiple queries to obtain the average of gradient estimates. When combined with Adam optimizer, MUZO-Adam demonstrates superior performance in fine-tuning various LLMs. Furthermore, we provide theoretical guarantees for the convergence of the MUZO-Adam optimizer. Extensive experiments empirically demonstrate that MUZO-Adam converges better than MeZO-SGD and achieves near first-order optimizer performance on downstream classification, multiple-choice, and generation tasks.

1 Introduction

In recent years, pre-training large language models (LLMs) on massive datasets followed by fine-tuning on downstream tasks has become the mainstream paradigm in large model training, achieving state-of-the-art performance across various domains (Sanh et al., 2022; Li et al., 2023; Wu et al., 2023). As model sizes grow, the substantial memory required for fine-tuning large models has become one of the primary challenges for downstream task adaptation. For instance, fully fine-tuning a 7B-parameter Llama-2 (Touvron et al., 2023) using the Adam (Kingma, 2014) optimizer requires at least 70GB of memory, which can easily exceed the memory limits of consumer GPUs.

The popular solution for addressing memory limitations is Parameter-Efficient Fine-Tuning (PEFT) (Lester et al., 2021), which encompasses techniques like Prefix-Tuning (Li and Liang, 2021) and Low-Rank Adaptation (LoRA) (Hu et al., 2021). While these methods fine-tune less than 1% of the model's parameters using first-order (FO) gradient optimizers such as Stochastic Gradient Descent (SGD) (Amari, 1993) and Adam, the GPU memory required for fine-tuning can still exceed multiple times the memory needed for inference. This is because the activation values must be stored during gradient computation for back-propagation.

To further reduce memory consumption during the fine-tuning of LLMs, the Memory-Efficient Zeroth-Order Optimization (MeZO) (Malladi et al., 2023) has been proposed. MeZO estimates gradients through two forward passes, avoiding back-propagation and thus eliminating the need to store activation values. When combined with the SGD optimizer, MeZO-SGD (Malladi et al., 2023) achieves fine-tuning of LLMs using only inference memory. However, ZO-SGD directly updates model parameters using the current step's gradient estimate, leading to issues such as unstable performance and slow convergence (Liu et al., 2020; Gautam et al., 2024). ZO-Adam leverages historical first and second moments to smooth gradient estimates and update steps, and has shown performance superior to ZO-SGD in some studies (Chen et al., 2019; Jiang et al., 2024). Nevertheless, limited by the significant noise in gradient estimates, moments in Adam also exhibit considerable noise, which restricts the performance of ZO-Adam.

In this paper, we propose the MUZO method, which efficiently obtains complete gradient estimates from multiple queries through layer-wise random seeds, thereby reducing the noise in gradient estimates. MUZO can be flexibly integrated

*Corresponding authors are Xie Chen and Fei Wen.

Proceedings of the 2025 Conference on Empirical Methods in Natural Language Processing, pages 8578–8596
November 4-9, 2025

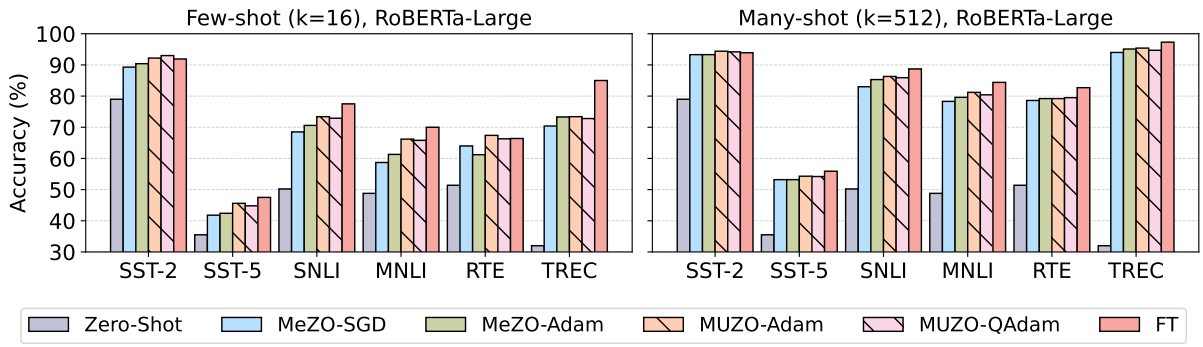

Figure 1: Experiments on RoBERTa-Large (350M parameters), with Zero-shot, MeZO-SGD, MeZO-Adam, MUZO-Adam, MUZO-QAdam and fine-tuning with Adam (FT). MUZO-Adam and MUZO-QAdam consistently outperform other methods and approaches in FT performance. More experimental results are shown in Table 1.

with momentum-based optimizers (e.g., Adam and SGD-Momentum) to improve convergence speed and performance. Through extensive theoretical analysis and experiments, we demonstrate that MUZO outperforms 1-query MeZO and LoZO (Malladi et al., 2023; Chen et al., 2024) in terms of performance and convergence when combined with various optimizers. In summary, our key contributions are listed below.

- We propose the MUZO method. Unlike traditional methods with multiple queries, our MUZO utilizes layer-wise independent perturbations and enables the efficient accumulation of gradient estimates over q queries, while incurring minimal additional overhead. MUZO is also extensible to MeZO variants.

- We develop the MUZO-Adam optimizer, which reduces the variance of gradient estimates through multiple queries and accelerates convergence by momentum, thus improving the performance of ZO fine-tuning. We propose MUZO-QAdam, a quantized method to further reduce the memory overhead of the Adam optimizer.

- We provide theoretical guarantees for the convergence of MUZO-Adam and conduct extensive experiments on LLMs, including RoBERTa-Large, OPT, Llama, and Vicuna (Liu et al., 2019; Zhang et al., 2022; Touvron et al., 2023; Chiang et al., 2023), demonstrating the superior performance of MUZO-Adam and MUZO-QAdam.

2 Related Work

2.1 Zeroth-order Optimization

Zeroth-order (ZO) optimization methods estimate gradients through finite differences of function values without relying on back-propagation. Extensive research on the convergence properties of ZO optimization in both convex and non-convex settings (Duchi et al., 2015; Jamieson et al., 2012; Wang et al., 2020; Ji et al., 2019) has been developed, demonstrating that the convergence depends on the number of parameters d. ZO optimization has been widely applied in adversarial attack and defense, automated machine learning (Ilyas et al., 2018; Tu et al., 2019; Wang et al., 2022), particularly in small-scale models. More recently, it has been introduced to the fine-tuning of LLMs, showcasing its potential in more complex machine learning problems (Malladi et al., 2023; Liu et al., 2024; Zhang et al., 2024).

2.2 Memory-efficient Fine-tuning

As the number of model parameters increases, memory-efficient fine-tuning of LLMs and foundation models has become an emerging challenge. Parameter Efficient Fine-Tuning (PEFT) (Lester et al., 2021) method was introduced to reduce the memory required for fine-tuning by minimizing the number of parameters to be updated. For example, LoRA (Hu et al., 2021) applies low-rank decomposition to the weight matrices, achieving performance close to full parameter fine-tuning while training only a small number of parameters. MeZO (Malladi et al., 2023), on the other hand, introduces zeroth-order optimization into LLMs fine-tuning, further reducing GPU memory requirements by avoiding the need to cache activations. Besides,

researchers in (Zhao et al., 2024; Hao et al., 2024; Chen et al., 2024) utilize low-rank subspace gradients to compress the memory needed to store the optimizer state.

3 Preliminary

This section presents MeZO (Malladi et al., 2023) and its low-rank variant LoZO (Chen et al., 2024). Both methods fall under the class of Randomized Vector-wise Gradient Estimation (RGE) approaches (Duchi et al., 2015; Nesterov and Spokoiny, 2017; Liu et al., 2018), which estimate gradients by applying a perturbation $\boldsymbol{z}$ to the model parameters:

$$\hat{\nabla}\mathcal{L}(\boldsymbol{\theta}) := \frac{\mathcal{L}(\boldsymbol{\theta}+\epsilon\boldsymbol{z})-\mathcal{L}(\boldsymbol{\theta}-\epsilon\boldsymbol{z})}{2\epsilon}\boldsymbol{z}, \quad (1)$$

where $\mathcal{L}$ is the loss function, and ϵ is the perturbation scale. $\boldsymbol{\theta} = \{\theta_i\}_{i=1}^{I}$ represents the set of model weight matrices, with θ_i denoting the weight matrix of the i-th layer.

3.1 Memory efficient ZO-SGD

MeZO (Malladi et al., 2023) was initially proposed for memory-efficient fine-tuning of LLMs. Unlike standard ZO-SGD, MeZO-SGD leverages a random seed trick to sample and regenerate perturbation vectors $z \sim \mathcal{N}(0,1)$ in Equation (1) using the same random seed, thereby eliminating the need to store perturbation vectors. This reduces the required memory to half that of standard ZO-SGD, enabling inference memory fine-tuning. The parameters' update rule is:

$$\boldsymbol{\theta}^{t+1} := \boldsymbol{\theta}^{t} - \eta\hat{\nabla}\mathcal{L}(\boldsymbol{\theta}^{t})$$

for learning rate η and model parameter $\boldsymbol{\theta}^t$ at t.

LoZO (Chen et al., 2024) is a variant algorithm of MeZO and uses a different perturbation sampling method. LoZO exploits the low-rank structure of FO gradients generated during the backpropagation of LLMs (Zhao et al., 2024; Hao et al., 2024). For a weight matrix $\theta_i \in \mathbb{R}^{m_i \times n_i}$, by sampling two low-rank random matrices $U_i \in \mathbb{R}^{m_i \times r_i}$ and $W_i \in \mathbb{R}^{n_i \times r_i}$ with rank $r_i \ll \min\{m_i, n_i\}$ from a normal distribution, it ensures that the random perturbation vector $z = U_i W_i^\top$ retains a low-rank property. The combination of LoZO with momentum methods requires storing a random matrix U_i for each weight matrix. Since the random matrix U_i is much smaller than z, LoZO can be memory-efficiently integrated with momentum techniques.

3.2 Multiple Queries RGE (q-RGE)

An extension of the RGE-based gradient estimation method (Duchi et al., 2015) is q-RGE, where the gradient is estimated independently using q queries, and the final gradient estimate is obtained by averaging the individual estimates:

$$\hat{\nabla}\mathcal{L}^{(q)}(\boldsymbol{\theta}) := \frac{1}{q}\sum_{i=1}^{q}\frac{\mathcal{L}(\boldsymbol{\theta}+\epsilon\boldsymbol{z}_i)-\mathcal{L}(\boldsymbol{\theta}-\epsilon\boldsymbol{z}_i)}{2\epsilon}\boldsymbol{z}_i. \quad (2)$$

Some experiments (Zhang et al., 2024; Ren et al., 2024) have demonstrated that using multiple queries in q-RGE can improve the performance of single-query RGE methods. However, the multi-query methods in (Zhang et al., 2024) did not achieve memory efficiency, nor did they accelerate the training process for q-RGE. On the other hand, the implementations in (Ren et al., 2024) were conducted only on Vision Transformers (Dosovitskiy et al., 2020), and replaced standard Convolutional Neural Network (CNN) (Krizhevsky et al., 2012) layers with custom layers. These limitations restrict the scalability of the approach to more general deep learning models.

3.3 Adaptive Momentum Optimizer

Adaptive Momentum Optimizer (Kingma, 2014; Shazeer and Stern, 2018; Zhao et al., 2024) based on first-order gradients, is a widely adopted optimization method in machine learning, offering better convergence compared to SGD. Previous research under the 1-query setting has demonstrated that applying Adam optimizer to ZO gradients can accelerate convergence (Chen et al., 2019; Jiang et al., 2024). ZO-Adam typically uses two types of historical gradient information: first-order moment in Equation (3) and second-order moment in Equation (4).

The first-order moment smooths the gradients, providing a more stable estimate of the average gradient direction over recent iterations:

$$\boldsymbol{m}^t := \beta_1 \boldsymbol{m}^{t-1} + (1-\beta_1)\hat{\nabla}\mathcal{L}(\boldsymbol{\theta}). \quad (3)$$

The second-order moment accounts for the variance of gradients across parameter dimensions, which allows for more efficient smoothing of update steps for each dimension:

$$\boldsymbol{v}^t := \beta_2 \boldsymbol{v}^{t-1} + (1-\beta_2)\hat{\nabla}\mathcal{L}(\boldsymbol{\theta})^2, \quad (4)$$

β_1, β_2 are exponential decay rates for the moment estimates.

Algorithm 1 MUZO Method (Main Algorithm)

Input: parameters $\boldsymbol{\theta} \in \mathbb{R}^d$, loss $\mathcal{L} : \mathbb{R}^d \to \mathbb{R}$, step budget T, perturbation scale ϵ, learning rate η, q query step accumulation
for $t = 1, \ldots, T$ **do**
  `proj_grads, seeds` $\leftarrow [\,]$
  for $j = 1, \ldots, q$ **do**
    Sample random seed s
    # Calling Algorithm 2
    $\boldsymbol{\theta} \leftarrow$ `Perturb`$(\boldsymbol{\theta}, \epsilon,$ `seeds`$, j,$ `True`$)$
    $\ell^+ \leftarrow \mathcal{L}(\boldsymbol{\theta})$
    $\boldsymbol{\theta} \leftarrow$ `Perturb`$(\boldsymbol{\theta}, -2\epsilon,$ `seeds`$, j,$ `False`$)$
    $\ell^- \leftarrow \mathcal{L}(\boldsymbol{\theta})$
    $\boldsymbol{\theta} \leftarrow$ `Perturb`$(\boldsymbol{\theta}, \epsilon,$ `seeds`$, j,$ `False`$)$
    `proj_grad` $\leftarrow (\ell^+ - \ell^-)/2\epsilon$
    `proj_grads[j]` $\leftarrow$ `proj_grad`
  end for
  for $\theta_i \in \boldsymbol{\theta}$ **do**
    # Calling Algorithm 3
    `grad` $\leftarrow$ `Getgrad(proj_grads, seeds,` i`)`
    $\theta_i \leftarrow \theta_i - \eta \cdot$ `optimizer(grad)`
  end for
end for

Algorithm 2 Perturbation Subroutine (`Perturb`)

Input: $\boldsymbol{\theta}$, ϵ, `seeds`, index j, boolean `new_seed`
for $\theta_i \in \boldsymbol{\theta}$ **do**
  if `new_seed` **then**
    Sample and set random seed ss
    `seeds[j][i]` $\leftarrow ss$
  else
    Set random seed with `seeds[j][i]`
  end if
  $z \sim \mathcal{N}(0, 1)$
  $\theta_i \leftarrow \theta_i + \epsilon z$
end for

Algorithm 3 Gradient Computation Subroutine (`Getgrad`)

Input: `proj_grads`, `seeds`, index i
`grad` $\leftarrow 0$
for $j = 1, \ldots, q$ **do**
  Reset seed `seeds[j][i]`
  $z \sim \mathcal{N}(0, 1)$
  `grad` $\leftarrow$ `grad` $+ z \cdot$ `proj_grads[j]`
end for
`grad` $\leftarrow$ `grad`$/q$
return `grad`

This paper extends this idea to the q-query settings, showing that integrating Adam with multiple-query ZO gradient estimation (MUZO-Adam) can further enhance performance by leveraging both variance reduction and adaptive momentum.

4 Methodology

In this section, we introduce **MUZO**, a novel **MU**ltiple-query, memory-efficient **Z**eroth-**O**rder method designed to reduce the variance in gradient estimation. Building upon the MUZO method, we further propose MUZO-Adam and establish its theoretical guarantees.

4.1 Proposed MUZO Method

At each training step t, the MUZO method consists of two phases: (1) **Gradient Estimation Phase**: The model undergoes random perturbations and forward passes in a cyclical order across q queries, caching the random seeds and gradient projection values. (2) **Weight Update Phase**: The model reconstructs the gradient estimate for each weight matrix in a cyclical order corresponding to the weight matrices and updates the weights accordingly.

MUZO provides a complete gradient estimate in a time- and memory-efficient manner. In the first perturbation phase of each step t, MUZO samples and stores the random seed for each query-layer pair, as shown in Algorithm 2. These random seeds are stored in a two-dimensional array `seeds`, which is used for resampling the perturbation vector in subsequent perturbation and gradient computation phases (Algorithm 3) with time complexity $O(1)$. As a result, MUZO can efficiently obtain the complete q-query gradient estimates for each weight matrix.

MUZO facilitates the efficient application of momentum-based optimizers in the q-query setting. As shown in Algorithm 1, the optimizer can be readily substituted with alternatives such as SGD-Momentum or Adam. In contrast, conventional q-SPSA MeZO algorithms (Malladi et al., 2023) require significantly larger memory caches or increased cycle times to acquire the complete q-query gradient estimates, thereby restricting them primarily to SGD optimizers. This limitation is further discussed in Appendix C.1.

4.2 MUZO-Adam

MUZO-Adam is introduced as the optimizer of the MUZO method combined with Adam. Differing from the 1-query ZO-Adam optimizer provided in Equation (3) and (4), MUZO-Adam applies the average gradient over multiple queries to estimate the gradient, instead of using a single query. Formally, at iteration t, MUZO-Adam estimates the gradient as in Equation (2), and first- and second-order moments are updated as follows:

$$\begin{aligned} \boldsymbol{m}^t &\leftarrow \beta_1 \boldsymbol{m}^{t-1} + (1-\beta_1)\hat{\nabla}\mathcal{L}^{(q)}(\boldsymbol{\theta}), \\ \boldsymbol{v}^t &\leftarrow \beta_2 \boldsymbol{v}^{t-1} + (1-\beta_2)\hat{\nabla}\mathcal{L}^{(q)}(\boldsymbol{\theta})^2. \end{aligned} \tag{5}$$

Followed by a coordinate-wise update:

$$\begin{aligned} \hat{\boldsymbol{V}}^t &= \operatorname{diag}\left(\max(\boldsymbol{v}^t, \boldsymbol{v}^{t-1})\right), \\ \boldsymbol{\theta}^{t+1} &= \boldsymbol{\theta}^t - \eta \hat{\boldsymbol{V}}^{-1/2t} \boldsymbol{m}^t, \end{aligned} \tag{6}$$

where $\hat{V}^t$ is the normalization matrix at iteration t. The detailed integration of the MUZO method with the Adam optimizer is presented in Appendix A.1.

MUZO-QAdam is introduced to further mitigate the optimizer memory overhead during full parameter fine-tuning. This approach involves quantizing the optimizer states required by Adam (i.e., the first and second moments) to FP8 precision, while the model weight perturbations and forward passes retain FP16/FP32 format. After the high-precision moments $\boldsymbol{m}^t$ and $\boldsymbol{v}^t$ are computed, they are converted to FP8 representations ($\boldsymbol{m}^t_{\text{FP8}}$ and $\boldsymbol{v}^t_{\text{FP8}}$) for storage. This process explicitly includes clipping the values to the representable range of the FP8 data type: $\boldsymbol{x}^t_{\text{FP8}} = \texttt{FP8Quantize}\left(\text{clamp}\left(\boldsymbol{x}^t, R^{\text{FP8}}_{\text{min}}, R^{\text{FP8}}_{\text{max}}\right)\right)$, where clamp$(\cdot)$ constrains its input to the range defined by $R^{\text{FP8}}_{\text{min}}$ and $R^{\text{FP8}}_{\text{max}}$, and $\texttt{FP8Quantize}(\cdot)$ performs the conversion to the FP8 data type. These FP8-stored moments are then de-quantized back to the FP16/FP32 precision in the subsequent calculation steps.

The underlying insight is that since the zeroth-order gradient estimation inherently introduces additional noise into the optimizer states, the first and second moments provide only an approximate direction for weight and step updates, thus obviating the need for high precision for these states. Besides, when applying MUZO-Adam to fine-tune the low-rank parameters introduced by methods such as LoRA, the memory footprint of the optimizer states is significantly reduced, amounting to less than 3% of the memory needed for inference.

Detailed MUZO-Adam and MUZO-QAdam algorithms are in Appendix A.1.

4.3 Theoretical Analysis

We begin by establishing that the gradient estimator is unbiased and has reduced variance.

Lemma 4.1. *Let $\hat{\nabla}\mathcal{L}^{(q)}(\boldsymbol{\theta})$ be the gradient estimate defined by Equation (2):*

$$\mathbb{E}[\hat{\nabla}\mathcal{L}^{(q)}(\boldsymbol{\theta})] = \nabla\mathcal{L}(\boldsymbol{\theta}^t). \tag{7}$$

This lemma ensures that $\hat{\nabla}\mathcal{L}^{(q)}(\boldsymbol{\theta})$ is an unbiased estimator of the true gradient $\nabla\mathcal{L}(\boldsymbol{\theta}^t)$ at iteration t.

Assumptions. We assume following conditions:

- **A1:** Function $\mathcal{L}(\boldsymbol{\theta})$ $\mathcal{L}$-smooth, for all $\boldsymbol{\theta}, \boldsymbol{\theta}' \in \mathbb{R}^d$, $\|\nabla\mathcal{L}(\boldsymbol{\theta}) - \nabla\mathcal{L}(\boldsymbol{\theta}')\|_2 \leq \mathcal{L}\|\boldsymbol{\theta} - \boldsymbol{\theta}'\|_2$.
- **A2:** Variance of stochastic gradient estimator $\hat{\nabla}\mathcal{L}^{(q)}(\boldsymbol{\theta})$ is bounded as $\frac{1}{B}\sum_{i=1}^{B}\|\nabla\mathcal{L}(\boldsymbol{\theta}) - \nabla\mathcal{L}(\boldsymbol{\theta}')\|_2^2 \leq \sigma^2$ for minibatch B.

Lemma 4.2. *The variance of $\hat{\nabla}\mathcal{L}^{(q)}(\boldsymbol{\theta})$ can be reduced by averaging over q queries. Specifically, for each iteration t:*

$$\mathbb{E}[\|\hat{\nabla}\mathcal{L}^{(q)}(\boldsymbol{\theta})\|_2^2] \leq \frac{\sigma^2}{q}. \tag{8}$$

Lemma 4.3. *The steady-state variances of the first and second moment estimates in MUZO-Adam, $\boldsymbol{m}^t$ and $\boldsymbol{v}^t$, are:*

$$\begin{aligned} V_{\boldsymbol{m}} &= \frac{1-\beta_1}{1+\beta_1}\frac{\sigma^2}{q}, \\ V_{\boldsymbol{v}} &= \frac{1-\beta_2}{1+\beta_2}\left(4\mu^2\frac{\sigma^2}{q} + 2\frac{\sigma^4}{q^2}\right), \end{aligned} \tag{9}$$

where μ is the mean gradient norm, β_1 and β_2 are coefficients of first and second moments, and σ^2 is the variance bound from Lemma 4.2. The variance of $\boldsymbol{m}^t$ decreases by a factor of q, while the variance of $\boldsymbol{v}^t$ includes terms proportional to $1/q$ and $1/q^2$.

Using these lemmas and assumptions, we establish the convergence result for MUZO-Adam:

Theorem 4.4. *Assume that **A1** and **A2** hold. Let $\{\boldsymbol{\theta}^t\}_{t=1}^T$ be the sequence generated by MUZO-Adam with q queries per iteration. Let $\boldsymbol{\theta}_R$ be chosen uniformly at random from $\{\boldsymbol{\theta}_1, \ldots, \boldsymbol{\theta}_T\}$, so*

$$\mathbb{E}[\|\nabla\mathcal{L}(\boldsymbol{\theta}_R)\|^2] = \mathcal{O}\left(\frac{d}{\sqrt{T}} + \frac{d^2}{T}\right), \tag{10}$$

where d is the dimension of the parameter space and T is the number of iterations.

Method	Few-Shot (k=16)						Many-Shot (k=512)					
	SST2	SST5	SNLI	MNLI	RTE	TREC	SST2	SST5	SNLI	MNLI	RTE	TREC
Zero-Shot	79.0	35.5	50.2	48.8	51.4	32.0	79.0	35.5	50.2	48.8	51.4	32.0
MeZO-SGD	90.3	42.8	68.5	58.7	64.0	70.4	93.3	53.2	83.0	78.3	78.6	94.0
MeZO-SGD-M	90.6	45.6	69.2	59.2	64.3	72.3	93.3	52.7	83.6	78.8	78.9	93.9
MeZO-Adam	90.4	42.4	70.6	62.3	62.2	73.3	93.3	53.2	85.3	79.6	79.2	95.1
LoZO-SGD	89.0	41.3	71.2	62.4	61.2	72.4	93.8	52.4	83.3	78.5	79.6	95.0
LoZO-SGD-M	88.0	42.9	71.0	62.7	60.2	**75.2**	94.3	52.6	84.9	79.5	**79.7**	95.2
MUZO-SGD	91.5	**46.1**	71.1	61.4	65.0	72.4	93.5	51.8	83.5	78.2	79.1	94.5
MUZO-Adam	92.2	45.6	**73.4**	**66.2**	**67.4**	73.4	**94.4**	**54.3**	**86.3**	**81.2**	79.2	**95.4**
MUZO-QAdam	**93.0**	44.8	72.9	65.8	66.3	72.8	94.2	54.2	85.9	80.4	79.5	94.7
FT	91.9	47.5	77.5	70.0	66.4	85.0	93.9	55.9	88.7	84.4	82.7	97.3

Table 1: Experiments on RoBERTa-Large (350M parameters). Our MUZO method is tested with different optimizers (SGD, SGD-Momentum, Adam, and QAdam). SGD-M refers to SGD-Momentum and FT means first-order fine-tuning with Adam optimizer. We re-ran all experiments to obtain a fair comparison, and all reported numbers are averaged results. Excluding FT in gray, the best results are shown in **bold**, the second best are shown in underline.

The detailed derivation of Lemma 4.2 and Lemma 4.3, and the proof of Theorem 4.4 are provided in Appendix A.2, A.3 and A.4.

5 Experiments

In this section, we evaluate the performance of MUZO by comparing it with other ZO methods, MeZO (Malladi et al., 2023) and LoZO (Chen et al., 2024), in combination with various optimizers, including SGD, SGD-Momentum, and Adam. We demonstrate through experiments that MUZO performs effectively across downstream classification, multiple-choice, and generation tasks.

Models. Our experiments evaluate the performance of MUZO across masked language models of RoBERTa-Large (Liu et al., 2019) and different sizes of autoregressive models, including OPT-13B, Llama-2-7B, and Vicuna-V1.5-7B (Zhang et al., 2022; Touvron et al., 2023; Chiang et al., 2023).

Datasets. Our experimental datasets cover various LLMs fine-tuning tasks, including most tasks in GLUE (Wang et al., 2018) and SuperGLUE (Wang et al., 2019) benchmarks. We adopt the experimental setup from (Malladi et al., 2023): for RoBERTa-Large, we study the few-shot (16 examples) and many-shot (512 examples) settings. For large autoregressive models, we randomly sample 1000, 500, and 1000 examples for training, validation, and test sets for each task.

Implementation Details. We set the same number of total forward passes, calculated by $2 \times q \times$ `Training_steps` for all experiments involving MUZO, MeZO and LoZO. For example, in the experiment with RoBERTa-Large, we set the number of training steps for MeZO to $100,000$, while for MUZO, the number of steps was set to $100,000/q$. FT steps are set to $1,000$ due to the much faster convergence. All the experiments are conducted on Nvidia Geforce RTX 3090 and A800 GPU. We provide specific prompts during fine-tuning for all experiments. More implementation details are in Appendix B.

5.1 Masked Language Models

Performance of MUZO consistently outperforms MeZO. In Table 1, MUZO combined with different optimizers achieves higher accuracy on most tasks and models, compared to 1-query MeZO, with this performance being more dominated under the few-shot learning setup. However, in the many-shot learning scenario, there are instances (e.g., SST-5 task) where MeZO-SGD outperforms MUZO-SGD. A possible reason is that, despite using a relatively higher learning rate to mitigate the undertraining caused by reduced training steps, MUZO-SGD still requires more forward passes to converge.

Acceleration provided by momentum compensates for the undertraining. Unlike the fluctuating performance of MUZO-SGD, MUZO-Adam consistently outperforms the 1-query method across both few-shot and many-shot settings. Figure 2 illustrates their loss curves, where it is evident that MUZO-Adam converges better, alleviating the dependency of MUZO on excessive forward passes,

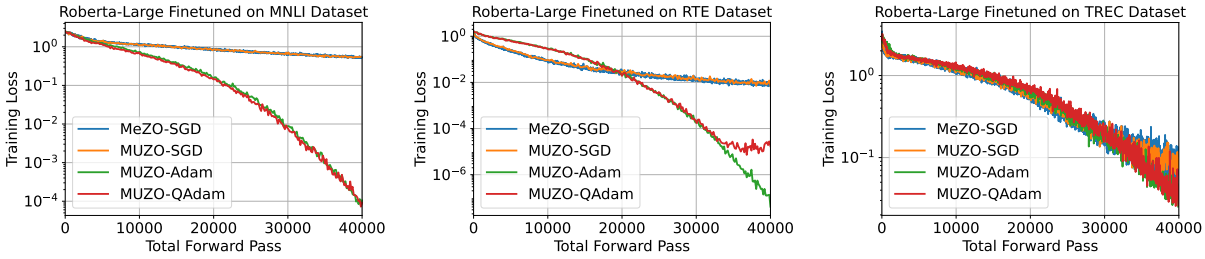

Figure 2: Training loss curves of RoBERTa-Large fine-tuned on MNLI, RTE, and TREC datasets.

Task	**SST2**	**RTE**	**CB**	**BoolQ**	**WSC**	**WIC**	**MultiRC**	**COPA**	**ReCoRD**	**SQuAD**	**DROP**
Task type	classification							– multiple choice –		— generation —	
Zero-Shot	58.8	59.6	46.4	59.0	38.5	55.0	46.9	80.0	81.0	46.2	14.6
MeZO-SGD	91.3	68.2	66.1	68.1	61.5	59.4	59.4	88.0	81.3	81.8	31.3
MeZO-SGD (LoRA)	89.6	67.9	67.8	73.5	**63.5**	60.2	61.3	84.0	81.5	82.1	31.3
LOZO	91.7	**70.4**	69.6	71.9	**63.5**	60.8	63.0	**89.0**	81.3	**82.9**	30.7
MUZO-Adam (LoRA)	**92.3**	69.7	**71.4**	**73.8**	63.3	**63.8**	**63.2**	88.0	**81.8**	**82.9**	**31.6**
FT	91.8	70.9	84.1	76.9	63.5	71.1	71.1	79.0	74.1	84.9	31.3

Table 2: Experiments on OPT-13B (with 1000 examples). MUZO-Adam achieved the best zero-order optimization performance on 8 out of 11 tasks. The best results are shown in **bold** except for FT in gray.

and enhancing its performance.

MUZO-Adam requires a slow warmup process to obtain the approximate direction of optimization, which is different from the fast convergence of FO-Adam. As shown in Figure 2, MUZO-Adam and MUZO-QAdam do not show obvious advantages over ZO-SGD in the early stage of training, and may even fall behind ZO-SGD (e.g., RTE task). However, after the number of training steps accumulates, ZO-Adam finds the optimization direction and the convergence speed significantly accelerates, while ZO-SGD relying on the gradient estimation of the current step shows slow convergence. Furthermore, the convergence speed of the MUZO-QAdam curve is almost identical to that of MUZO-Adam, only converging prematurely when the loss is less than $1e-5$ due to lower precision.

5.2 Autoregressive Language Models

MUZO-Adam demonstrates strong performance in classification, multiple-choice, and generation tasks when combined with LoRA. As shown in Tables 2 and 3, in experiments with large autoregressive models, we tested different models and datasets, leading to conclusions similar to those obtained with RoBERTa-Large. Specifically, MUZO-Adam achieved performance superior to both MeZO and LoZO on most tasks. Furthermore, due to the utilization of the LoRA method, MUZO-Adam requires almost no additional memory compared to ZO-SGD methods. Table 4 presents the GPU peak memory required by different optimization algorithms, showing that MUZO-Adam maintains its memory advantage.

	Llama2-7B		**Vicuna-7B**	
Method	COPA	Wino.	COPA	Wino.
MeZO-SGD	84.0	64.3	83.0	65.6
MeZO-SGD-M	84.0	64.5	83.0	64.2
MeZO-Adam	82.0	64.4	84.0	65.2
MUZO-SGD	85.0	64.0	**85.0**	65.8
MUZO-SGD-M	**86.0**	65.0	83.0	65.1
MUZO-Adam	85.0	**65.2**	**85.0**	**66.1**
FT	84.0	69.5	83.0	70.0

Table 3: Performance of Llama and Vicuna finetuned with LoRA on COPA and WinoGrande (Wino.) using different ZO optimizers. The best results are shown in **bold** except for FT in gray.

5.3 More Discussions

Variance Reduction. Figure 3 illustrates the loss curves under different numbers of queries. When the number of queries increases, the loss curve descends more smoothly, and the rate of descent also varies. We present the performance under different query numbers in the ablation studies. Under the same number of forward steps, increasing query numbers may improve performance. However, when the number of queries is too high, the number of epochs the model iterates through the dataset

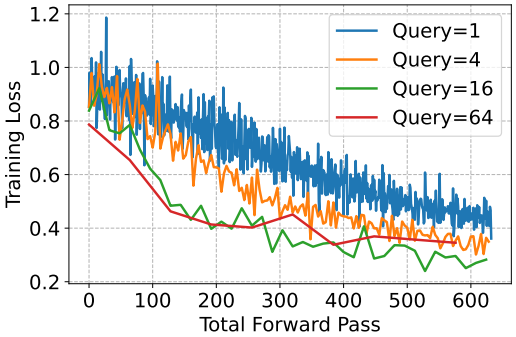

Figure 3: Training loss curve of OPT-13B finetuned on SST-2 dataset. The learning rates are scaled with query numbers.

Optimizer	w/ LoRA	w/o LoRA
FO-SGD	36.6 GB	67.4 GB
FO-Adam	36.8 GB	90.8 GB
MeZO-SGD	13.0 GB	13.6 GB
MUZO-Adam	13.1 GB	35.2 GB
MUZO-QAdam	13.0 GB	18.5 GB

Table 4: Memory consumption of MUZO, MeZO and FO method to finetune OPT-2.7B on SST-2 task.

decreases, leading to degraded performance.

Memory Efficiency. Table 4 shows the memory requirements of MUZO-Adam when choosing either LoRA fine-tuning or optimizer quantization. Compared to inference memory, using MUZO-Adam directly with full-parameter fine-tuning requires over 150% additional memory. However, after applying the quantized MUZO-QAdam optimizer, this overhead can be reduced to less than 50%. The most memory-efficient MUZO-Adam fine-tuning still requires combining with LoRA. In this setting, MUZO-Adam can achieve optimization of inference memory while obtaining performance superior to ZO-SGD. Additionally, using low-rank sampling methods in LoZO also has the potential to improve memory efficiency, as shown in Appendix D.

Computational Cost. Table 5 analyzes the GPU hours and convergence forward passes required for fine-tuning the RoBERTa on the SST2 task using MUZO. MUZO-Adam reduces GPU hours and total forward passes by 70% compared to MeZO, but still needs three times of GPU hours compared to FO-Adam.

5.4 Ablation Study

We conducted an ablation study on query numbers to investigate whether directly increasing the number of queries to q and scaling the learning rate by a factor of q could improve the performance of MUZO-Adam. The experimental results are shown in Table 6 and Figure 3. Scaling the learning rate directly achieves good performance when $q = 8$ or $q = 16$, but as q increases further, the performance may deteriorate due to excessively high learning rates and fewer epochs over the training set. For other ablation studies, such as the impact of Adam, QAdam and SGD optimizer on MUZO, we provide a comprehensive analysis directly in Table 1 and Figure 2.

Optimizer	Forward Pass	GPU Hour
MeZO-SGD	200K	21.3 (100%)
MUZO-QAdam	60K	6.51 (31%)
FO-Adam	4K	2.12 (10%)

Table 5: Computational cost of MUZO, MeZO, and FO method to finetune RoBERTa-Large on RTE task, with `batch_size` $= 64$. FO-Adam needs 4 steps of gradient accumulation to avoid out of memory, so the number of forward passes is 4 times of training steps.

Query	SNLI	RTE	TREC
q=1	70.6	62.2	73.3
q=2	71.3	63.5	73.1
q=4	71.9	66.4	73.3
q=8	**73.4**	**67.4**	73.2
q=16	72.6	65.8	**73.4**
q=32	72.1	66.0	72.9

Table 6: Ablation study on the number of queries in one training step to finetune RoBERTa-Large on three tasks. When the query number increases, training steps decrease because we fixed the same total forward passes.

6 Conclusion

This paper introduces MUZO, a novel zeroth-order optimization method that efficiently computes the multi-query gradient average, thereby reducing the variance of the gradient estimate. We propose the MUZO-Adam optimizer, which leverages the Adam optimizer to determine the optimization direction, thereby accelerating convergence. To mitigate the memory overhead associated with the Adam optimizer, we further introduce the MUZO-QAdam optimizer, which employs FP8 quantization. Extensive experiments demonstrate that both MUZO-Adam and MUZO-QAdam exhibit superior performance and convergence speed compared to other zeroth-order optimization baselines in both full-parameter and LoRA fine-tuning settings.

Acknowledgments

This work was supported by the Science and Technology Innovation (STI) 2030-Major Project (2022ZD0208700), National Natural Science Foundation of China (No. U23B2018 and No. 62206171), Shanghai Municipal Science and Technology Major Project under Grant 2021SHZDZX0102 and Yangtze River Delta Science and Technology Innovation Community Joint Research Project (2024CSJGG01100).

Limitations

Despite the performance and convergence improvements achieved by the proposed MUZO-Adam and MUZO-QAdam methods for zeroth-order optimization, there remains a gap when compared to the BP-based FO-Adam optimizer. MUZO-Adam still has an accuracy degradation of about 2% and requires at least 3 times the GPU hours compared to FO-Adam. Further enhancing the performance and convergence of the zeroth-order optimizer continues to be a challenging problem.

References

Shun-ichi Amari. 1993. Backpropagation and stochastic gradient descent method. *Neurocomputing*, 5(4-5):185–196.

Samuel Bowman, Gabor Angeli, Christopher Potts, and Christopher D Manning. 2015. A large annotated corpus for learning natural language inference. In *Proc. EMNLP*, pages 632–642.

Xiangyi Chen, Sijia Liu, Kaidi Xu, Xingguo Li, Xue Lin, Mingyi Hong, and David Cox. 2019. ZO-AdaMM: Zeroth-order adaptive momentum method for black-box optimization. *In Proc. NeurIPS*, 32:7202–7213.

Yiming Chen, Yuan Zhang, Liyuan Cao, Kun Yuan, and Zaiwen Wen. 2024. Enhancing zeroth-order finetuning for language models with low-rank structures. *arXiv preprint arXiv:2410.07698*.

Wei-Lin Chiang, Zhuohan Li, Zi Lin, Ying Sheng, Zhanghao Wu, Hao Zhang, Lianmin Zheng, Siyuan Zhuang, Yonghao Zhuang, Joseph E Gonzalez, and 1 others. 2023. Vicuna: An open-source chatbot impressing GPT-4 with 90%* ChatGPT quality. *See https://vicuna. lmsys. org*, 2(3):6.

Christopher Clark, Kenton Lee, Ming-Wei Chang, Tom Kwiatkowski, Michael Collins, and Kristina Toutanova. 2019. Boolq: Exploring the surprising difficulty of natural yes/no questions. *arXiv preprint arXiv:1905.10044*.

Sanjoy Dasgupta and Anupam Gupta. 2003. An elementary proof of a theorem of johnson and lindenstrauss. *Random Structures & Algorithms*, 22(1):60–65.

Marie-Catherine De Marneffe, Mandy Simons, and Judith Tonhauser. 2019. The commitmentbank: Investigating projection in naturally occurring discourse. In *proceedings of Sinn und Bedeutung*, volume 23, pages 107–124.

Alexey Dosovitskiy, Lucas Beyer, Alexander Kolesnikov, Dirk Weissenborn, Xiaohua Zhai, Thomas Unterthiner, Mostafa Dehghani, Matthias Minderer, Georg Heigold, Sylvain Gelly, and 1 others. 2020. An image is worth 16x16 words: Transformers for image recognition at scale. *arXiv preprint arXiv:2010.11929*.

John C Duchi, Michael I Jordan, Martin J Wainwright, and Andre Wibisono. 2015. Optimal rates for zero-order convex optimization: The power of two function evaluations. *IEEE Transactions on Information Theory*, 61(5):2788–2806.

Tanmay Gautam, Youngsuk Park, Hao Zhou, Parameswaran Raman, and Wooseok Ha. 2024. Variance-reduced zeroth-order methods for fine-tuning language models. *arXiv preprint arXiv:2404.08080*.

Yongchang Hao, Yanshuai Cao, and Lili Mou. 2024. FLORA: Low-rank adapters are secretly gradient compressors. *arXiv preprint arXiv:2402.03293*.

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

Andrew Ilyas, Logan Engstrom, Anish Athalye, and Jessy Lin. 2018. Black-box adversarial attacks with limited queries and information. In *Proc. ICML*, pages 2137–2146. PMLR.

Kevin G Jamieson, Robert Nowak, and Ben Recht. 2012. Query complexity of derivative-free optimization. *In Proc. NeurIPS*, 25:2681–2689.

Kaiyi Ji, Zhe Wang, Yi Zhou, and Yingbin Liang. 2019. Improved zeroth-order variance reduced algorithms and analysis for nonconvex optimization. In *Proc. ICML*, pages 3100–3109. PMLR.

Shuoran Jiang, Qingcai Chen, Youcheng Pan, Yang Xiang, Yukang Lin, Xiangping Wu, Chuanyi Liu, and Xiaobao Song. 2024. ZO-AdaMU Optimizer: Adapting perturbation by the momentum and uncertainty in zeroth-order optimization. In *Proc. AAAI*, volume 38, pages 18363–18371.

Daniel Khashabi, Snigdha Chaturvedi, Michael Roth, Shyam Upadhyay, and Dan Roth. 2018. Looking beyond the surface: A challenge set for reading comprehension over multiple sentences. In *Proceedings*

of the 2018 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies, Volume 1 (Long Papers), pages 252–262.

Diederik P Kingma. 2014. Adam: A method for stochastic optimization. *arXiv preprint arXiv:1412.6980*.

Alex Krizhevsky, Ilya Sutskever, and Geoffrey E Hinton. 2012. Imagenet classification with deep convolutional neural networks. *In Proc. NeurIPS*, 25:84–90.

Brian Lester, Rami Al-Rfou, and Noah Constant. 2021. The power of scale for parameter-efficient prompt tuning. In *Proc. EMNLP*, pages 3045–3059.

Hector J Levesque, Ernest Davis, and Leora Morgenstern. 2012. The winograd schema challenge. *KR*, 2012:13th.

Junnan Li, Dongxu Li, Silvio Savarese, and Steven Hoi. 2023. Blip-2: Bootstrapping language-image pre-training with frozen image encoders and large language models. In *Proc. ICML*, pages 19730–19742. PMLR.

Xiang Lisa Li and Percy Liang. 2021. Prefix-tuning: Optimizing continuous prompts for generation. In *Proc. IJCNLP*, pages 4582–4597.

Sijia Liu, Pin-Yu Chen, Bhavya Kailkhura, Gaoyuan Zhang, Alfred O Hero III, and Pramod K Varshney. 2020. 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.

Sijia Liu, Bhavya Kailkhura, Pin-Yu Chen, Paishun Ting, Shiyu Chang, and Lisa Amini. 2018. Zeroth-order stochastic variance reduction for nonconvex optimization. *In Proc. NeurIPS*, 31:3731–3741.

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

Yong Liu, Zirui Zhu, Chaoyu Gong, Minhao Cheng, Cho-Jui Hsieh, and Yang You. 2024. Sparse MeZO: Less parameters for better performance in zeroth-order llm fine-tuning. *arXiv preprint arXiv:2402.15751*.

Sadhika Malladi, Tianyu Gao, Eshaan Nichani, Alex Damian, Jason D Lee, Danqi Chen, and Sanjeev Arora. 2023. Fine-tuning language models with just forward passes. *In Proc. NeurIPS*, 36:53038–53075.

Yurii Nesterov and Vladimir Spokoiny. 2017. Random gradient-free minimization of convex functions. *Foundations of Computational Mathematics*, 17(2):527–566.

Mohammad Taher Pilehvar and Jose Camacho-Collados. 2018. Wic: the word-in-context dataset for evaluating context-sensitive meaning representations. *arXiv preprint arXiv:1808.09121*.

Pranav Rajpurkar, Jian Zhang, Konstantin Lopyrev, and Percy Liang. 2016. Squad: 100,000+ questions for machine comprehension of text. *arXiv preprint arXiv:1606.05250*.

Tao Ren, Zishi Zhang, Jinyang Jiang, Guanghao Li, Zeliang Zhang, Mingqian Feng, and Yijie Peng. 2024. Flops: Forward learning with optimal sampling. *arXiv preprint arXiv:2410.05966*.

Victor Sanh, Albert Webson, Colin Raffel, Stephen Bach, Lintang Sutawika, Zaid Alyafeai, Antoine Chaffin, Arnaud Stiegler, Arun Raja, Manan Dey, and 1 others. 2022. Multitask prompted training enables zero-shot task generalization. In *Proc. ICLR*.

Noam Shazeer and Mitchell Stern. 2018. Adafactor: Adaptive learning rates with sublinear memory cost. In *Proc. ICML*, pages 4596–4604. PMLR.

Richard Socher, Alex Perelygin, Jean Wu, Jason Chuang, Christopher D Manning, Andrew Y Ng, and Christopher Potts. 2013. Recursive deep models for semantic compositionality over a sentiment treebank. In *Proc. EMNLP*, pages 1631–1642.

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

Chun-Chen Tu, Paishun Ting, Pin-Yu Chen, Sijia Liu, Huan Zhang, Jinfeng Yi, Cho-Jui Hsieh, and Shin-Ming Cheng. 2019. AutoZOOM: Autoencoder-based zeroth order optimization method for attacking black-box neural networks. In *Proc. AAAI*, volume 33, pages 742–749.

Alex Wang, Yada Pruksachatkun, Nikita Nangia, Amanpreet Singh, Julian Michael, Felix Hill, Omer Levy, and Samuel Bowman. 2019. Superglue: A stickier benchmark for general-purpose language understanding systems. *In Proc. NeurIPS*, 32:3261–3275.

Alex Wang, Amanpreet Singh, Julian Michael, Felix Hill, Omer Levy, and Samuel Bowman. 2018. Glue: A multi-task benchmark and analysis platform for natural language understanding. In *Proc. EMNLP*, pages 353–355.

Xiaoxing Wang, Wenxuan Guo, Jianlin Su, Xiaokang Yang, and Junchi Yan. 2022. Zarts: On zero-order optimization for neural architecture search. *In Proc. NeurIPS*, 35:12868–12880.

Zhongruo Wang, Krishnakumar Balasubramanian, Shiqian Ma, and Meisam Razaviyayn. 2020. Zeroth-order algorithms for nonconvex minimax problems with improved complexities. *arXiv preprint arXiv:2001.07819*.

Adina Williams, Nikita Nangia, and Samuel Bowman. 2018. A broad-coverage challenge corpus for sentence understanding through inference. In *Pro. ACL*, pages 1112–1122.

Jian Wu, Yashesh Gaur, Zhuo Chen, Long Zhou, Yimeng Zhu, Tianrui Wang, Jinyu Li, Shujie Liu, Bo Ren, Linquan Liu, and 1 others. 2023. On decoder-only architecture for speech-to-text and large language model integration. In *2023 Proc. ASRU*, pages 1–8. IEEE.

Sheng Zhang, Xiaodong Liu, Jingjing Liu, Jianfeng Gao, Kevin Duh, and Benjamin Van Durme. 2018. Record: Bridging the gap between human and machine commonsense reading comprehension. *arXiv preprint arXiv:1810.12885*.

Susan Zhang, Stephen Roller, Naman Goyal, Mikel Artetxe, Moya Chen, Shuohui Chen, Christopher Dewan, Mona Diab, Xian Li, Xi Victoria Lin, and 1 others. 2022. Opt: Open pre-trained transformer language models. *arXiv preprint arXiv:2205.01068*.

Yihua Zhang, Pingzhi Li, Junyuan Hong, Jiaxiang Li, Yimeng Zhang, Wenqing Zheng, Pin-Yu Chen, Jason D Lee, Wotao Yin, Mingyi Hong, and 1 others. 2024. Revisiting zeroth-order optimization for memory-efficient LLM fine-tuning: A benchmark. *arXiv preprint arXiv:2402.11592*.

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

A MUZO-Adam

A.1 MUZO-Adam Structure

We present the MUZO method integrated with the Adam and QAdam optimizer, referred to as *MUZO-Adam* and *MUZO-QAdam*. The gradient is estimated over multiple queries, and the moment updates are carried out using the Adam rules, as shown in Algorithm 4.

A.2 Proof of Lemma 4.2

To prove Lemma 4.2, we need to show that the variance of the gradient estimator $\hat{\nabla}\mathcal{L}^{(q)}(\boldsymbol{\theta})$ can be reduced by averaging over q queries. The gradient estimator $\hat{\nabla}\mathcal{L}^{(q)}(\boldsymbol{\theta})$ is defined as the average of q independent stochastic gradient estimates, where each estimates $\hat{g}_t$ is calculated using a random perturbation at iteration t. The gradient estimator $\hat{\nabla}\mathcal{L}^{(q)}(\boldsymbol{\theta})$ is the average of q independent random gradient estimates:

$$\hat{\nabla}\mathcal{L}^{(q)}(\boldsymbol{\theta}) = \frac{1}{q}\sum_{i=1}^{q}\hat{g}_i. \tag{11}$$

Each $\hat{g}_i$ is a random variable depending on the random perturbation applied to $\boldsymbol{\theta}$. Since $\hat{g}_i$ is a gradient estimate, we can assume that its expectation is the true gradient $\nabla\mathcal{L}(\boldsymbol{\theta})$. Therefore, $\mathbb{E}[\hat{g}_i] = \nabla\mathcal{L}(\boldsymbol{\theta})$.

We also know that the variance of each gradient estimate $\hat{g}_i$ is bounded, and it is given by Assumption A2. The variance of the gradient estimate $\hat{g}_i$ is $\mathbb{E}[\|\hat{g}_i - \nabla\mathcal{L}(\boldsymbol{\theta})\|_2^2] \leq \sigma^2$.

The variance of the averaged estimator $\hat{\nabla}\mathcal{L}^{(q)}(\boldsymbol{\theta})$ can be computed by using the fact that the gradient estimates $\hat{g}_i$ are independent $\mathbb{E}[\|\hat{\nabla}\mathcal{L}^{(q)}(\boldsymbol{\theta})\|_2^2] = \mathbb{E}\left[\left\|\frac{1}{q}\sum_{i=1}^{q}\hat{g}_i\right\|_2^2\right]$.

Since the $\hat{g}_i$ are independent, we can expand the expectation:

$$\begin{aligned}\mathbb{E}\left[\left\|\frac{1}{q}\sum_{i=1}^{q}\hat{g}_i\right\|_2^2\right] =& \frac{1}{q^2}\sum_{i=1}^{n}\mathbb{E}[\|\hat{g}_i\|_2^2] \\ &+ \frac{1}{q^2}\sum_{i\neq j}\mathbb{E}[\langle\hat{g}_i,\hat{g}_j\rangle].\end{aligned} \tag{12}$$

Since the $\hat{g}_i$ are independent, the cross terms vanish, $\mathbb{E}[\langle\hat{g}_i,\hat{g}_j\rangle] = 0 \quad \text{for} \quad i \neq j$. Thus, the expression simplifies to: $\mathbb{E}\left[\left\|\frac{1}{q}\sum_{i=1}^{q}\hat{g}_i\right\|_2^2\right] = \frac{1}{q^2}\sum_{i=1}^{q}\mathbb{E}[\|\hat{g}_i\|_2^2]$.

Now, using the fact that $\hat{g}_i$ is an unbiased estimator with bounded variance, we know that the expectation of the squared norm of each $\hat{g}_i$ is bounded by σ^2, we have $\mathbb{E}[\|\hat{g}_i\|_2^2] \leq \sigma^2$. Therefore, variance of the averaged gradient estimator is bounded by:

$$\mathbb{E}\left[\left\|\frac{1}{q}\sum_{i=1}^{q}\hat{g}_i\right\|_2^2\right] = \frac{1}{q^2}\cdot q\cdot\sigma^2 = \frac{\sigma^2}{q}. \quad (13)$$

The derivation of Lemma 4.2 builds upon the mathematical framework and techniques introduced in ZO-SVRG (Liu et al., 2018), which similarly employs multiple perturbation vectors, consistent with our settings.

A.3 Proof of Lemma 4.3

Following the notation of Appendix A.2, we derive the variances of the moment estimates when the process would reach a steady state, where $\text{Var}(\boldsymbol{m}^t) = \text{Var}(\boldsymbol{m}^{t-1}) = V_{\boldsymbol{m}}$ and $\text{Var}(\boldsymbol{v}^t) = \text{Var}(\boldsymbol{v}^{t-1}) = V_{\boldsymbol{v}}$.

The first moment estimate is updated as $\boldsymbol{m}^t = \beta_1\boldsymbol{m}^{t-1} + (1-\beta_1)\hat{g}_t^{(q)}$, where $\hat{g}_t^{(q)} = \hat{\nabla}\mathcal{L}^{(q)}(\boldsymbol{\theta})$.

The variance of the first moment estimate is:

$$\text{Var}(\boldsymbol{m}^t) = \beta_1^2\text{Var}(\boldsymbol{m}^{t-1}) + (1-\beta_1)^2\text{Var}(\hat{g}_t^{(q)}). \quad (14)$$

By Lemma 4.2, and at steady state, so:

$$V_{\boldsymbol{m}} = \beta_1^2 V_{\boldsymbol{m}} + (1-\beta_1)^2\frac{\sigma^2}{q}. \quad (15)$$

Solving that:

$$V_{\boldsymbol{m}} = \frac{1-\beta_1}{1+\beta_1}\frac{\sigma^2}{q}. \quad (16)$$

For MeZO-Adam, its variance is $\frac{1-\beta_1}{1+\beta_1}\sigma^2$, which is q times larger than for MUZO-Adam.

On the other hand, the second moment estimate is updated as $\boldsymbol{v}^t = \beta_2\boldsymbol{v}^{t-1} + (1-\beta_2)(\hat{g}_t^{(q)})^2$. The variance is:

$$\text{Var}(\boldsymbol{v}^t) = \beta_2^2\text{Var}(\boldsymbol{v}^{t-1}) + (1-\beta_2)^2\text{Var}\left((\hat{g}_t^{(q)})^2\right). \quad (17)$$

The variance of the squared gradient estimate is:

$$\text{Var}\left((\hat{g}_t^{(q)})^2\right) = \mathbb{E}\left[(\hat{g}_t^{(q)})^4\right] - \left(\mathbb{E}\left[(\hat{g}_t^{(q)})^2\right]\right)^2. \quad (18)$$

Assuming $\mu = \|\mathbb{E}[\hat{g}_t^{(q)}]\|_2$, we approximate the fourth moment, yielding:

$$\text{Var}\left((\hat{g}_t^{(q)})^2\right) \approx 4\mu^2\frac{\sigma^2}{q} + 2\frac{\sigma^4}{q^2}. \quad (19)$$

At steady state, so:

$$V_{\boldsymbol{v}} = \beta_2^2 V_{\boldsymbol{v}} + (1-\beta_2)^2\left(4\mu^2\frac{\sigma^2}{q} + 2\frac{\sigma^4}{q^2}\right). \quad (20)$$

Solving that:

$$V_{\boldsymbol{v}} = \frac{1-\beta_2}{1+\beta_2}\left(4\mu^2\frac{\sigma^2}{q} + 2\frac{\sigma^4}{q^2}\right). \quad (21)$$

For MeZO-Adam, it variance is $\frac{1-\beta_2}{1+\beta_2}\left(4\mu^2\sigma^2 + 2\sigma^4\right)$, where both terms are larger by factors of q and q^2, respectively, demonstrating the quadratic noise reduction in MUZO-Adam.

A.4 Proof of Theorem 4.4

In this section, we establish the convergence bounds for MUZO-Adam based on Equations 5 and 6. Referring to (Chen et al., 2019), we present the following proposition:

Proposition A.1. *Suppose that assumptions A1 and A2 hold, and let $\boldsymbol{\theta} = \mathbb{R}^d$. Let the parameters be set as follows: $\sqrt{\hat{V}_0} \geq c$, $L_\mu(\boldsymbol{\theta}_1) - \min_{\boldsymbol{\theta}} L_\mu(\boldsymbol{\theta}) \leq D_f$, $\gamma := \beta_1/\beta_2 < 1$, $\mu = 1/\sqrt{Td}$. Then, the following convergence bound holds:*

$$\begin{aligned}\mathbb{E}\left[\left\|\hat{V}_R^{-1/4}\nabla\mathcal{L}(\boldsymbol{\theta}_R)\right\|^2\right] &\leq \frac{L_g^2 d}{2cT} + \frac{2D_f\sqrt{d}}{\sqrt{T}} \\ &+ \frac{L_g(4+5\beta_1^2)(1-\beta_1)}{2(1-\beta_1)^2(1-\beta_2)(1-\gamma)}\frac{\sqrt{d}}{\sqrt{T}} \\ &+ \frac{2}{c}\mathbb{E}\left[2\eta^2 + \frac{\eta\max_{t\in[T]}\|\hat{g}_t\|_\infty}{1-\beta_1}\right]\frac{d}{T},\end{aligned} \quad (22)$$

where $\boldsymbol{\theta}_R$ is selected uniformly at random from $\{\boldsymbol{\theta}_t\}_{t=1}^T$, and $\hat{g}_t = \hat{\nabla}\mathcal{L}(\boldsymbol{\theta}_t)$.

Proposition A.1 indicates that the convergence rate of MUZO-Adam depends on the zero-order gradient estimates, particularly on the quantity $G_{zo} := \max_{t\in[T]}\|\hat{g}_t\|_\infty$. In the ZO setting (Chen et al., 2019), this quantity is influenced by the dimensionality d, as demonstrated by the bound:

$$\begin{aligned}\|\hat{\nabla}\mathcal{L}(\boldsymbol{\theta})\|_2 &\leq (d/\mu)\|\mathcal{L}(\boldsymbol{\theta}+\mu) - \mathcal{L}(\boldsymbol{\theta})\|_2 \\ &\leq dL_c,\end{aligned} \quad (23)$$

where $\{\mathcal{L}(\boldsymbol{\theta})\}$ is assumed to satisfy the L_c-Lipschitz continuity condition under Assumption A3. From Equation (11), each $\hat{g}_i$ represents an independent zero-order gradient estimate computed using random perturbations. While averaging over q queries reduces the variance of the gradient estimator (as shown in Lemma 4.2), it does not directly influence the individual values of $\|\hat{g}_t\|_\infty$, as

the maximum of the individual gradient estimates remains bounded by function's Lipschitz constant L_c.

Proposition A.2 refines this dimensional dependency of G_{zo} by utilizing sphere concentration results (Chen et al., 2019), as stated below:

Proposition A.2. *Under the assumption of L_c-Lipschitz continuity for $\{\mathcal{L}(\boldsymbol{\theta})\}$ and given $\delta \in (0,1)$, with probability at least $(1-\delta)$,*

$$\max_{t\in[T]} \|\hat{g}_t\|_\infty \leq 2L_c\sqrt{d\log(dT/\delta)}. \tag{24}$$

This result provides a tighter bound for the gradient estimate norm, demonstrating that $\|\hat{g}_t\|_\infty$ scales as $\mathcal{O}(\sqrt{d})$ with high probability, instead of linearly with d. The dependency on T accounts for the increasing likelihood of extreme gradient estimates over time.

Proof of Proposition A.2 Define $G_{zo,i} := \max_{t\in[T]} |\hat{g}_{t,i}|$. By Lemma 2.2 of (Dasgupta and Gupta, 2003), for a vector $\mathbf{u}$ sampled uniformly from the unit sphere in $\mathbb{R}^d$, we have:

$$P(|u_i| \geq \sqrt{\xi/d}) \leq \exp\left((1-\xi+\log\xi)/2\right) \tag{25}$$

Let $\xi = 4\log\frac{dT}{\delta}$. Using the assumption $\max(d,T) \geq 3$, we get $P(|u_i| \geq \sqrt{\xi/d}) \leq \delta/dT$.

Recall that ZO gradient estimate $\hat{g}_t$ is given by

$$\hat{\nabla}\mathcal{L}(\boldsymbol{\theta}) := \frac{d}{\epsilon}[\mathcal{L}(\boldsymbol{\theta}+\epsilon\boldsymbol{z}) - \mathcal{L}(\boldsymbol{\theta})]\boldsymbol{z}. \tag{26}$$

By Lipschitz continuity of $\{\mathcal{L}(\boldsymbol{\theta})\}$ under Assumption **A2**, the i-th coordinate of $\hat{g}_{t,i}$ is upper bounded by $dL_c|u_i|$. Since $\boldsymbol{u}$ is drawn uniformly randomly from a unit sphere and $|\hat{g}_{t,i}| \leq dL_c|u_i|$, we have

$$\begin{aligned} &P(dL_c|u_i| \geq L_c\sqrt{\xi d}) \leq \delta/dT \\ \implies &P(|\hat{g}_{t,i}| \geq L_c\sqrt{\xi d}) \leq \delta/dT. \end{aligned} \tag{27}$$

Substituting $\xi = 4\log\frac{dT}{\delta}$ yields

$$P(|\hat{g}_{t,i}| \geq 2L_c\sqrt{d\log(dT/\delta)}) \leq \delta/dT. \tag{28}$$

Finally, applying the union bound over all $i \in [d], t \in [T]$, we obtain

$$\begin{aligned} &P\left[|\hat{g}_{t,i}| \geq 2L_c\sqrt{d\log(dT/\delta)}, \forall i,t\right] \\ &\leq \sum_{t\in[T]}\sum_{i\in[d]} P\left[|\hat{g}_{t,i}| \geq 2L_c\sqrt{d\log(dT/\delta)}\right] \\ &\leq dT(\delta/dT) = \delta, \end{aligned} \tag{29}$$

which proves Proposition A.2.

Using Propositions A.1 and A.2, we derive the following convergence rate for MUZO-Adam (Chen et al., 2019):

Theorem A.3. *Suppose that assumptions A1 and A3 hold. Given the parameter settings specified in Propositions A.1 and A.2, with probability at least $1-1/(T\sqrt{d})$, MUZO-Adam satisfies*

$$\mathbb{E}\left[\left\|\hat{V}_R^{-1/4}\nabla\mathcal{L}(\boldsymbol{\theta}_R)\right\|^2\right] = \mathcal{O}\left(\sqrt{d}/\sqrt{T} + d^{1.5}/T\right) \tag{30}$$

We can further refine the convergence rate of MUZO-Adam using the measure $\mathbb{E}[\|\nabla\mathcal{L}(\boldsymbol{\theta}_R)\|^2]$. Noting that $\hat{V}_t^{-1/2} \geq 1/\max_{t\in[T]}\|\hat{g}_t\|_\infty$ (due to the update rule), we derive:

$$\begin{aligned} \mathbb{E}[\|\nabla\mathcal{L}(\boldsymbol{\theta}_R)\|^2] \leq &2L_c\sqrt{d\log(dT/\delta)} \\ &\mathbb{E}\left[\left\|\hat{V}_R^{-1/4}\nabla\mathcal{L}(\boldsymbol{\theta}_R)\right\|^2\right]. \end{aligned} \tag{31}$$

From this, we obtain an upper bound on the expected squared gradient norm in terms of the algorithm parameters, concluding that the convergence rate of MUZO-Adam is $\mathcal{O}(d/\sqrt{T} + d^2/T)$ under standard evaluation measures.

B Experimental Details

B.1 Datasets

The datasets for experiment of RoBERTa-Large include SST-2, SST-5, SNLI, MNLI, RTE and TREC (Socher et al., 2013; Bowman et al., 2015; Williams et al., 2018).

For autoregressive language models, we conduct experiments on the following datasets: SST-2, RTE, CB, BoolQ, WSC, WIC, MultiRC, COPA, ReCoRD, SQuAD, DROP (Socher et al., 2013; De Marneffe et al., 2019; Clark et al., 2019; Levesque et al., 2012; Pilehvar and Camacho-Collados, 2018; Khashabi et al., 2018; Zhang et al., 2018; Rajpurkar et al., 2016).

B.2 Hyperparameters Selection

For the fine-tuning experiment on RoBERTa-Large and OPT-13B, we performed a grid search over the following hyperparameters. We set a relatively higher learning rate for MUZO compared to MeZO. To control for the same number of forward passes, in the RoBERTa-Large experiment, we set the training steps for MeZO to $100,000$ and for MUZO to $100,000/q$. All the FT experiments run $1,000$ training steps.

Experiment	Hyperparameters	Values
MeZO	Batch size	$\{64\}$
	Learning rate	$\{1e-6, 3e-7\}$
	ϵ	$\{3e-3, 1e-3\}$
LoZO	Batch size	$\{64\}$
	Learning rate	$\{1e-6, 3e-7\}$
	ϵ	$\{3e-3, 1e-3\}$
MUZO	Batch size	$\{64\}$
	Learning rate	$\{3e-6, 1e-6\}$
	ϵ	$\{3e-3, 1e-3\}$
	q	$\{4, 8, 16\}$
FT	Batch size	$\{8\}$
	Learning rate	$\{1e-5, 5e-5\}$

Table 7: Hyperparameter grids used for RoBERTa-Large experiments.

Experiment	Hyperparameters	Values
MeZO	Batch size	$\{24\}$
	Learning rate	$\{1e-6, 3e-7\}$
	ϵ	$\{1e-3\}$
LOZO	Batch size	$\{24\}$
	Learning rate	$\{1e-6, 3e-6\}$
	ϵ	$\{1e-3\}$
MUZO (LoRA)	Batch size	$\{24\}$
	Learning rate	$\{3e-4, 1e-4\}$
	ϵ	$\{3e-3, 1e-3\}$
	q	$\{4, 8, 16\}$
FT	Batch size	$\{8\}$
	Learning rate	$\{1e-4, 1e-5\}$

Table 8: Hyperparameter grids used for OPT-13B experiments.

In the OPT-13B experiment, we set the training steps for MeZO to $20,000$ and for MUZO to $20,000/q$.

C More Discussions about q-SPSA MeZO

C.1 q-SPSA MeZO

To facilitate readability, we present the complete q-SPSA MeZO in Algorithm 5.

We decompose the q-SPSA MeZO method into two parts: The first part is responsible for obtaining gradient estimates, which include q-query as the outer loop and three perturbations as the inner loop, used to perturb the parameter weight matrix layer by layer. The second part focuses on updating the model weights using the acquired gradient estimates, which also include q-query as the outer loop and a single perturbation as the inner loop. In the second part, all parameter update operations are performed within the q-query loop, and in-place operators are used to save memory, avoiding the need to cache the gradient estimates for the $j-th$ query. Therefore, in q-SPSA MeZO, the sum of q gradient estimates for each weight matrix cannot be directly obtained, and thus prevents the computation of moments in ZO-Adam optimizer.

Two straightforward approaches exist, but both have significant drawbacks. (1) One approach is to cache the accumulated gradients for all weight matrices. However, this incurs an additional memory overhead equivalent to the size of the fine-tuning parameters. (2) Another solution is to introduce a third nested weight matrix loop outside the existing two, caching the gradient of the selected weight matrix. While this reduces memory usage, it increases the time complexity of parameter updates by a factor of N, where N is the number of perturbed weight matrices.

C.2 Trick of Random Seeds

A random seed is a value used to initialize a pseudo-random number (or vector) generator. When reinitialized with the same seed, the generator will produce the same sequence of random numbers. In the context of the q-SPSA MeZO method, this means that by initializing the pseudo-random vector generator with a specific random seed before each iteration of the weight matrices loop, we can regenerate the same random vector sequence, thereby reproducing all random perturbations z and avoiding the memory cache for z itself.

However, reproducing the sequence requires strictly adhering to the order in which the random vectors are generated. For instance, it is not possible to skip the random vector generation for θ_i and directly generate the random vector for θ_{i+1}, nor can we generate directly the random vector for θ_{i-1} after θ_i has already been generated. All random vectors must be generated sequentially. Therefore, within the q-SPSA MeZO method, obtaining a specific random perturbation z for a weight matrix θ_i requires first generating the random perturbations for all preceding layers (i.e., $\theta_1, \theta_2, \ldots, \theta_{i-1}$). The time complexity of this process is $O(N)$, where N is the number of weight matrices.

In contrast, the MUZO method reduces this time complexity to $O(1)$, as it caches the random seeds for each weight matrix, allowing for the direct generation of random vectors without needing to traverse previous layers.

D MUZO Extensions

In this section, we present the combination of MUZO and LoZO, referred to as the low-rank MUZO method. Some more straightforward variants of the MeZO method, such as Sparse-MeZO(Liu et al., 2024), can be directly implemented by modifying the sampling of $z \sim \mathcal{N}(0, 1)$ by an additional mask.

LoZO method with 1-query can achieve memory efficiency by caching U_i and W_i without relying on random seed reproduction, because the sizes of low-rank matrices $U_i \in \mathbb{R}^{m_i \times r_i}$ and $W_i \in \mathbb{R}^{n_i \times r_i}$ are much smaller than the size of full rank matrix $z \sim \mathcal{N}(0, 1) \in \mathbb{R}^{m_i \times n_i}$. However, when the number of queries is large, i.e. $q \cdot r_i \not\ll \min(m_i, n_i)$, the caching method still leads to a significant increase in memory usage. Therefore, for the frequently sampled vector U_i, we adopt the random seed trick, while for the lazily sampled vector W_i, we retain the caching method.

Algorithms 6, 7, and 8 present the complete low-rank MUZO method. The implementation of low-rank MUZO-Momentum requires only the addition of the momentum update process in the *Getgrad function* (Algorithm 8) as well as the projection of momentum in the *Main Algorithm* (Algorithm 6).

$$M_i \leftarrow \beta M_i + (1 - \beta) \cdot \texttt{proj\_grad[j]} \cdot U_i, \quad (32)$$

where M_i is the momentum for weight matrix θ_i in low-rank subspace and β is the momentum factor.

Algorithm 4 MUZO-Adam and MUZO-QAdam

Input: parameters $\boldsymbol{\theta} \in \mathbb{R}^d$, loss $\mathcal{L} : \mathbb{R}^d \to \mathbb{R}$, step budget T, perturbation scale ϵ, learning rate η, q query step accumulation, $\beta_1, \beta_2 \in (0, 1]$, $opti\_type$

for $\theta_i \in \boldsymbol{\theta}$ **do**
  $m_i, v_i \leftarrow 0$ *# Initialize first- and second-order moments*
end for
for $t = 1, \ldots, T$ **do**
  $\texttt{proj\_grads}, \texttt{seeds} \leftarrow [\,]$
  for $j = 1, \ldots, q$ **do**
    Sample random seed s
    $\boldsymbol{\theta} \leftarrow \texttt{Perturb}(\boldsymbol{\theta}, \epsilon, \texttt{seeds}, j, \texttt{True})$ *# Calling Algorithm 2*
    $\ell^+ \leftarrow \mathcal{L}(\boldsymbol{\theta})$
    $\boldsymbol{\theta} \leftarrow \texttt{Perturb}(\boldsymbol{\theta}, -2\epsilon, \texttt{seeds}, j, \texttt{False})$
    $\ell^- \leftarrow \mathcal{L}(\boldsymbol{\theta})$
    $\boldsymbol{\theta} \leftarrow \texttt{Perturb}(\boldsymbol{\theta}, \epsilon, \texttt{seeds}, j, \texttt{False})$
    $\texttt{proj\_grad} \leftarrow (\ell^+ - \ell^-)/2\epsilon$
    $\texttt{proj\_grads[j]} \leftarrow \texttt{proj\_grad}$
  end for
  for $\theta_i \in \boldsymbol{\theta}$ **do**
    $\texttt{grad} \leftarrow \texttt{Getgrad}(\texttt{proj\_grads}, \texttt{seeds}, i)$ *# Calling Algorithm 3*
    $\theta_i \leftarrow \texttt{AdamOptimizer}(\texttt{grad}, \boldsymbol{m}, \boldsymbol{v}, \eta, i, opti\_type)$ *# Integrated with Adam optimizer*
  end for
end for

Subroutine $\texttt{AdamOptimizer}(\texttt{grad}, \boldsymbol{m}, \boldsymbol{v}, \eta, i, opti\_type)$
if $\texttt{opti\_type}$ = "QAdam" **then**
  $m_i, v_i \leftarrow DeQuantize(m_i, v_i)$
end if
$m_i \leftarrow \beta_1 m_i + (1 - \beta_1) \times \texttt{grad}$
$v_i \leftarrow \beta_2 v_i + (1 - \beta_2) \times \texttt{grad}^2$
$\hat{v}_i \leftarrow v_i/(1 - {\beta_2}^t)$
$\theta_i \leftarrow \theta_i - \eta \hat{v}_i^{-1/2} m_i$
if $\texttt{opti\_type}$ = "QAdam" **then**
  $m_i, v_i \leftarrow FP8QuantizeWithClipping(m_i, v_i)$
end if
return θ_i

Algorithm 5 q-SPSA MeZO

Input: parameters $\boldsymbol{\theta} \in \mathbb{R}^d$, loss $\mathcal{L} : \mathbb{R}^d \to \mathbb{R}$, step budget T, perturbation scale ϵ, learning rate η, q query step accumulation

for $t = 1, \ldots, T$ **do**
  $\texttt{seeds}, \texttt{proj\_grads}, \leftarrow [\,]$
  # First q-query loop
  for $j = 1, \ldots, q$ **do**
    Sample random seed s
    $\boldsymbol{\theta} \leftarrow \texttt{Perturb}(\boldsymbol{\theta}, \epsilon, s)$ *# Weight matrices loop*
    $\ell^+ \leftarrow \mathcal{L}(\boldsymbol{\theta})$
    $\boldsymbol{\theta} \leftarrow \texttt{Perturb}(\boldsymbol{\theta}, -2\epsilon, s)$ *# Weight matrices loop*
    $\ell^- \leftarrow \mathcal{L}(\boldsymbol{\theta})$
    $\boldsymbol{\theta} \leftarrow \texttt{Perturb}(\boldsymbol{\theta}, \epsilon, s)$ *# Weight matrices loop*
    $\texttt{proj\_grad} \leftarrow (\ell^+ - \ell^-)/(2\epsilon)$
    $\texttt{proj\_grads[j]} \leftarrow \texttt{proj\_grad}$
    $\texttt{seeds[j]} \leftarrow s$
  end for
  # Second q-query loop
  for $j = 1, \ldots, q$ **do**
    Reset seed $\texttt{seeds[j]}$ *# Weight matrices loop*
    for $\theta_i \in \boldsymbol{\theta}$ **do**
      $z \sim \mathcal{N}(0, 1)$
      $\theta_i \leftarrow \theta_i - (\eta/q) * \texttt{proj\_grad[j]} * z$
    end for
  end for
end for

Subroutine $\texttt{Perturb}(\boldsymbol{\theta}, \epsilon, s)$
Set random seed s
for $\theta_i \in \boldsymbol{\theta}$ **do**
  $z \sim \mathcal{N}(0, 1)$
  $\theta_i \leftarrow \theta_i + \epsilon z$
end for

Algorithm 6 Low-Rank MUZO Method (Main Algorithm)

Input: parameters $\boldsymbol{\theta} \in \mathbb{R}^d$, loss $\mathcal{L} : \mathbb{R}^d \to \mathbb{R}$, step budget T, perturbation scale ϵ, learning rate η, q query step accumulation, sample interval w and rank $\{r_i\}$.
for $\theta_i \in \boldsymbol{\theta}$ **do**
  `W_list` $\leftarrow [\,]$
end for
for $t = 1, \ldots, T$ **do**
  `proj_grads`, `seeds` $\leftarrow [\,]$
  for $j = 1, \ldots, q$ **do**
    Sample random seed s
    if t mod w = 0 **then**
      for $\theta_i \in \boldsymbol{\theta}$ **do**
        Sample $W_i \in \mathbb{R}^{n_i \times r_i}$
        `W_list`$[i] \leftarrow W_i$
      end for
    end if
    $\boldsymbol{\theta} \leftarrow$ `Perturb`$(\boldsymbol{\theta}, \epsilon,$ `seeds`, `V_list`$, j,$ `True`$)$ *# Calling Algorithm 7*
    $\ell^+ \leftarrow \mathcal{L}(\boldsymbol{\theta})$
    $\boldsymbol{\theta} \leftarrow$ `Perturb`$(\boldsymbol{\theta}, -2\epsilon,$ `seeds`, `V_list`$, j,$ `False`$)$
    $\ell^- \leftarrow \mathcal{L}(\boldsymbol{\theta})$
    $\boldsymbol{\theta} \leftarrow$ `Perturb`$(\boldsymbol{\theta}, \epsilon,$ `seeds`, `V_list`$, j,$ `False`$)$
    `proj_grad` $\leftarrow (\ell^+ - \ell^-)/2\epsilon$
    `proj_grads`[j] $\leftarrow$ `proj_grad`
  end for
  for $\theta_i \in \boldsymbol{\theta}$ **do**
    `grad` $\leftarrow$ `Getgrad`$($`proj_grads`, `seeds`, `V_list`$, i)$ *# Calling Algorithm 8*
    $\theta_i \leftarrow \theta_i - \eta \cdot$ `optimizer`$($`grad`$)$
  end for
end for

Algorithm 7 Low-Rank MUZO Perturbation Subroutine (`Perturb`)

Input: $\boldsymbol{\theta}$, ϵ, `seeds`, `W_list`, index j, boolean `new_seed`
for $\theta_i \in \boldsymbol{\theta}$ **do**
  if `new_seed` **then**
    Sample and set random seed ss
    `seeds`[j][i] $\leftarrow ss$
  else
    Set random seed with `seeds`[j][i]
  end if
  Sample $U_i \sim \mathcal{N}(0, 1) \in \mathbb{R}^{m_i \times r_i}$
  $W_i \leftarrow$ `W_list`$[i]$
  $\theta_i \leftarrow \theta_i + \epsilon U_i W_i^\top$
end for

Algorithm 8 Low-Rank MUZO Gradient Computation Subroutine (`Getgrad`)

Input: `proj_grads`, `seeds`, `W_list`, index i
`grad` $\leftarrow 0$
for $j = 1, \ldots, q$ **do**
  Reset seed `seeds`[`j`][`i`]
  Sample $U_i \sim \mathcal{N}(0, 1) \in \mathbb{R}^{m_i \times r_i}$
  $W_i \leftarrow$ `W_list`$[i]$
  `grad` $\leftarrow$ `grad` $+ proj\_grads[j] \cdot U_i W_i^\top / r_i$
end for
`grad` $\leftarrow$ `grad`$/q$
return `grad`