Harmony in Divergence: Towards Fast, Accurate, and Memory-efficient Zeroth-order LLM Fine-tuning

Qitao Tan [1] **Jun Liu** [2] **Zheng Zhan** [2] **Caiwen Ding** [3] **Yanzhi Wang** [2] **Jin Lu** [1] **Geng Yuan** [1]

Abstract

Large language models (LLMs) excel across various tasks, but standard first-order (FO) fine-tuning demands considerable memory, significantly limiting real-world deployment. Recently, zeroth-order (ZO) optimization stood out as a promising memory-efficient training paradigm, avoiding backward passes and relying solely on forward passes for gradient estimation, making it attractive for resource-constrained scenarios. However, ZO method lags far behind FO method in both convergence speed and accuracy. To bridge the gap, we introduce a novel layer-wise divergence analysis that uncovers the distinct update pattern of FO and ZO optimization. Aiming to resemble the learning capacity of FO method from the findings, we propose **Di**vergence-driven **Z**eroth-**O**rder (**DiZO**) optimization. DiZO conducts divergence-driven layer adaptation by incorporating projections to ZO updates, generating diverse-magnitude updates precisely scaled to layer-wise individual optimization needs. Our results demonstrate that DiZO significantly reduces the needed iterations for convergence without sacrificing throughput, cutting training GPU hours by up to 48% on various datasets. Moreover, DiZO consistently outperforms the representative ZO baselines in fine-tuning RoBERTa-large, OPT-series, and Llama-series on downstream tasks and, in some cases, even surpasses memory-intensive FO fine-tuning.

1. Introduction

Fine-tuning pre-trained large language models (LLMs) with backpropagation demonstrates superior performance for many natural language processing tasks (Yang et al., 2019; Liu et al., 2019; Talmor et al., 2018; Chowdhery et al., 2023; Zheng et al., 2020). However, the extensive parameterization imposes a substantial memory burden, limiting their practicality for widespread downstream applications. In line with the neural scaling laws (Hoffmann et al., 2022; Kaplan et al., 2020), next-generation LLMs continue to increase in parameter count. Specifically, model sizes are expanding at a rate of 410× every two years, dramatically outpacing the scaling of DRAM bandwidth (1.4× every two years) and DRAM capacity (2× every two years). This disparity leads to the *memory wall* challenge (Gholami et al., 2024), which becomes even more severe when deploying LLMs on memory-limited devices (Zeng et al., 2024; Chen et al., 2024; Hur et al., 2023).

Recently, zeroth-order (ZO) optimization has emerged as a promising memory-efficient training paradigm for LLM fine-tuning, attracting significant attention (Zhang et al., 2024; Liu et al., 2024; Malladi et al., 2023; Zhao et al., 2024). By relying solely on forward passes (i.e., inference) to estimate gradients and update model parameters, ZO bypasses the need for backward propagation and significantly reduces extensive storage requirements for activations, gradients, and optimizer states. As reported in Malladi et al. (2023), fine-tuning LLMs via ZO optimization reduces up to 12× memory cost. Nevertheless, ZO optimization still exhibits a **gap** in convergence speed and accuracy compared to the conventional first-order (FO) method (i.e., compute gradient via backpropagation). As shown in Table 1, one can observe that the FO method substantially outperforms ZO method in both accuracy and GPU hours. Though ZO method achieves higher throughput due to its computational simplicity, it requires more than 10× iterations for convergence, dramatically increasing GPU hours. Previous studies typically attribute this gap to the fact that ZO optimization leverages random perturbation for gradient estimation, and thus results in unavoidable estimation error, but without further exploration of other underlying causes (Malladi et al., 2023; Gautam et al., 2024; Zhao et al., 2024).

To bridge this gap, we begin by examining the distinct update patterns shown by ZO and FO methods during LLM fine-tuning. Our analysis reveals a substantial difference in their layer-wise update magnitudes. Specifically, ZO method relies on high-dimensional random search and tends to apply uniform-magnitude updates without consid-

[1]University of Georgia [2]Northeastern University [3]University of Minnesota Twin Cities. Correspondence to: Geng Yuan <geng.yuan@uga.edu>.

ering layer-wise individual characteristics. In contrast, FO method benefits from fine-grained gradient estimation and applies diverse-magnitude updates precisely scaled to the layer-wise individual optimization needs. Motivated by these observations, we are interested in investigating: *if we can also provide ZO with diverse-magnitude updates, effectively achieving training acceleration and accuracy improvement.*

Drawing on these insights, we propose **Di**vergence-driven **Z**eroth-**O**rder optimization (**DiZO**). DiZO conducts divergence-driven layer adaptation by incorporating projections, enabling layer-wise adaptive updates that closely resemble FO approaches. Notably, the projections can be optimized without gradients, ensuring that DiZO retains the appealing backpropagation-free features. Moreover, we validate DiZO on a variety of tasks, including classification and generation, using several LLMs such as RoBERTa-large, the OPT series, and the Llama series. Experimental results show that DiZO substantially decreases training iterations for convergence while maintaining throughput, cutting training GPU hours by up to 48% on diverse datasets. Furthermore, our method can be seamlessly integrated with parameter-efficient tuning techniques like low-rank adaptation (Hu et al., 2021) for additional speedups. DiZO also consistently outperforms the representative ZO baselines and, in some cases, surpasses memory-intensive FO fine-tuning.

The summary of our contributions is as follows:

- We introduce a novel layer-wise divergence analysis to uncover the fundamental differences in the updating patterns of FO and ZO methods.
- We introduce DiZO, a novel ZO method using divergence-driven layer adaptation, achieving a learning capacity closely resembling FO while maintaining the throughput benefit of ZO optimization.
- DiZO consistently exceeds existing baselines in both accuracy and GPU hours, and it can be seamlessly integrated with LoRA for additional benefits. These advantages hold across diverse tasks and LLM architectures.

2. Preliminaries and Pattern Analysis

2.1. Revisiting Zeroth-order Optimization

Recently, ZO optimization has gained significant attention in machine learning (Verma et al., 2023; Dhurandhar et al., 2019; Wang et al., 2022; Gu et al., 2021). Unlike conventional FO optimization, which calculates gradients via backpropagation, ZO optimization estimates gradients using only objective oracles via finite differences (Chen et al., 2023; Liu et al., 2018; Ye et al., 2018). This property can be leveraged for LLM fine-tuning to alleviate the extensive memory costs. Specifically, as ZO only needs two forward passes to obtain the estimated gradients, it avoids computing and storing the most memory-consuming information needed in the conventional FO training, i.e., activations in the forward process, gradients in the backward process, and the optimizer state.

Table 1. Fine-tuning results on SST-2 datasets. Although ZO method shows advantages in memory saving, left behind FO method in terms of both accuracy and GPU hours.

Model	Type	Acc.	Memory	#Train Iter.	GPU Hours
RoBERTa	FO	91.9	9.2 GB	6.6%	12.3%
	ZO	90.5	4.5 GB	100.0%	100.0%
OPT-2.7B	FO	94.2	45.4 GB	7.5%	16.8%
	ZO	90.0	6.8 GB	100.0%	100.0%

The core idea of ZO optimization is to estimate gradients by applying random perturbations to the weights and computing differences in the objective. For a mini-batch of data $\mathcal{B}$, sampled from a labeled dataset $\mathcal{D} = \{x_i, y_i\}_{i=1}^{|\mathcal{D}|}$, a model with parameters $\boldsymbol{\theta} \in \mathbb{R}^d$, where d represents the dimension of the parameter space, and the corresponding loss function $\mathcal{L}(\boldsymbol{\theta}; \mathcal{B})$. The gradient is estimated as follows:

$$\nabla \mathcal{L}(\boldsymbol{\theta}; \mathcal{B}) = \frac{1}{q} \sum_{i=1}^{q} \left[\frac{\mathcal{L}\left(\boldsymbol{\theta} + \epsilon \boldsymbol{u}_i; \mathcal{B}\right) - \mathcal{L}\left(\boldsymbol{\theta} - \epsilon \boldsymbol{u}_i; \mathcal{B}\right)}{2\epsilon} \boldsymbol{u}_i \right] \tag{1}$$

where $\boldsymbol{u}_i$ is a random vector with the same dimension as the model weights and is typically drawn from standard Gaussian distribution $\mathcal{N}(0, \mathbf{I})$ (Malladi et al., 2023; Zhang et al., 2024), or from Gaussian sampling over a unit sphere (Liu et al., 2018; Shamir, 2017), q is the number of objective queries, and $\epsilon > 0$ is a small perturbation scalar for smoothing.

Given the learning rate η and the mini-batch data $\mathcal{B}_t$ at t-th iteration, once the estimated gradient $\nabla \mathcal{L}(\theta; \mathcal{B}_t)$ is obtained, then ZO-SGD updates the parameters with the following rule:

$$\boldsymbol{\theta}_{t+1} = \boldsymbol{\theta}_t - \eta \nabla \mathcal{L}(\boldsymbol{\theta}_t; \mathcal{B}_t) \tag{2}$$

2.2. Layer-wise Divergence Analysis

Drawing insight from the update formula of ZO optimization, we notice that ZO method applies uniform-magnitude updates across layers, e.g., the L2-norm of the updates is about the same for all layers in one iteration (see Appendix E for proof). This fact may be the root of the inferior performance of ZO optimization. To measure how the divergence of update magnitude affects the convergence speed and accuracy, we investigate the training dynamics of ZO and FO methods, respectively.

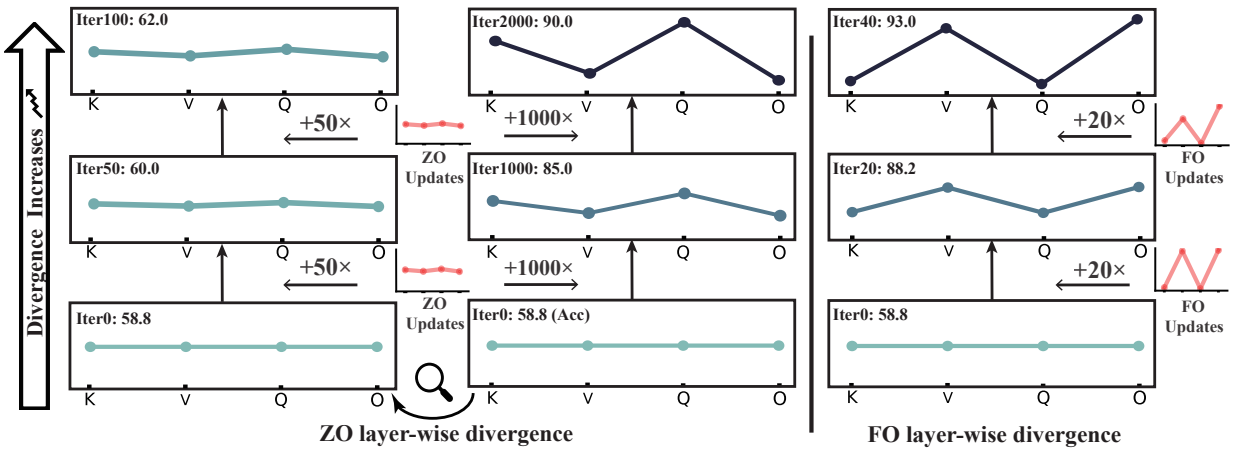

Figure 1. Comparison of the training dynamics of ZO and FO methods. The X-axis represents layer names, and the Y-axis represents the distance gap. Although they converge to different stable states, the divergence of the distance gap increases in both FO and ZO methods during training. FO accumulates divergence rapidly through diverse-magnitude updates, while ZO applies uniform-magnitude updates, requiring more iterations for an ideal divergence level. Results are obtained by fine-tuning OPT-2.7B on the SST-2 dataset, focusing on weights in the attention module: K (Key), V (Value), Q (Query), and O (Output projection).

Analysis indicator. To quantify the effect of updates, we adapt the layer-wise L2-norm distance gap between the weights of the pre-trained and the fine-tuned model as an indicator. The layer-wise L2-norm distance gap is defined as:

$$\|\Delta\boldsymbol{\theta}_t^{(\ell)}\| = \|\boldsymbol{\theta}_t^{(\ell)} - \boldsymbol{\theta}_0^{(\ell)}\|_2 \tag{3}$$

where t is t-th fine-tuning iteration, ℓ is ℓ-th layer of the model, and $\boldsymbol{\theta}_0^{(\ell)}$ indicates the weights of ℓ-th layer of pre-trained model.

Analysis result. Figure 1 compares the training dynamics of FO and ZO methods. Regardless of whether ZO or FO is used, the divergence of distance gap among layers grows during training, i.e., the line of distance gap gradually 'bends'. This pattern implies that different layers benefit from maintaining diverse distance gaps with the pre-trained model. However, FO and ZO diverge in how the distance gap divergence is accumulated. FO employs fine-grained gradient estimations, resulting in diverse-magnitude updates (FO updates in Figure 1). Therefore, it can rapidly reach the desired layer-wise distance gap in only a few iterations. In contrast, ZO relies on random search in high-dimensional parameter space and generates uniform-magnitude updates (ZO updates in Figure 1), resulting in thousands more iterations required for accumulating a meaningful layer-wise distance gap.

With the above findings, we suspect the inferior performance of ZO stems from its inability to deliver layer-wise adaptive updates, a challenge that arises from its reliance on random perturbations for gradient estimation.

Algorithm 1 Divergence-diven ZO Optimization (DiZO)

Require: parameter of t-th iteration $\boldsymbol{\theta_t}$ and pre-trained model $\boldsymbol{\theta_0}$, loss function $\mathcal{L}$, step budget T, perturbation scalar ϵ, mini-batch data $\mathcal{B}_t$, learning rate η, projection at t-th iteration $\boldsymbol{\gamma_t} = \{\gamma_t^i\}_{i=1}^L$

for $t = 1$ **to** T **do**
  $\nabla\mathcal{L} = \texttt{GradEst}(\boldsymbol{\theta_t}, \epsilon, \mathcal{B}_t)$
  $\boldsymbol{\theta_t} = \boldsymbol{\theta_{t-1}} - \eta\nabla\mathcal{L}$
  $\boldsymbol{\gamma}_t^* = \arg\min_{\boldsymbol{\gamma}_t} \mathcal{L}(\boldsymbol{\theta}_0 + \frac{\boldsymbol{\gamma}_t}{\|\Delta\boldsymbol{\theta}_t\|}\Delta\boldsymbol{\theta}_t; \mathcal{B}_t)$
  $\boldsymbol{\theta}_t = \texttt{ApplyProjection}(\boldsymbol{\theta}_t, \boldsymbol{\theta}_0, \boldsymbol{\gamma}_t^*)$
end
Subroutine `GradEst` $(\boldsymbol{\theta}, \epsilon, \mathcal{B})$**:**
  Sample: $u_1, \ldots, u_q \backsim \mathcal{N}(0, \mathbf{I})$
  Query: $y_i = \mathcal{L}(\boldsymbol{\theta} + \epsilon u_i; \mathcal{B}) - \mathcal{L}(\boldsymbol{\theta} - \epsilon u_i; \mathcal{B})$
  Estimator: $\nabla\mathcal{L} = \frac{q}{2\epsilon}\sum_{i=1}^q y_i u_i$
  return $\nabla\mathcal{L}$
return
Subroutine `ApplyProjection` $(\boldsymbol{\theta}_t, \boldsymbol{\theta}_0, \boldsymbol{\gamma}_t)$**:**
  for $\ell = 1, 2, \ldots, L$ **do**
    `// Project` l`-th layer`
    $\boldsymbol{\theta}_t^{(\ell)} = \boldsymbol{\theta}_0^{(\ell)} + \frac{\boldsymbol{\gamma}_t^{(\ell)}}{\|\Delta\boldsymbol{\theta}_t^{(\ell)}\|}\Delta\boldsymbol{\theta}_t^{(\ell)}$
  end
  return $\boldsymbol{\theta}_t$
return

3. Methodology

We find that ZO applies uniform-magnitude updates for all layers, which could be the root of its inferior performance in accuracy and convergence speed. Consequently, we introduce a variant of ZO optimization which performs divergence-driven layer adaptation, thereby providing diverse-magnitude updates to enhance the overall learning capacity.

3.1. Design of the Divergence-driven Layer Adaptation

To provide layer-wise adaptive updates for ZO optimization, we apply projections to the updates of different layers, generating updates with diverse magnitudes. The pseudocode for the proposed method is shown in Algorithm 1.

Specifically, We treat training iteration as a two-step process that iteratively updates the weights and the projection factor. Our approach involves two key steps performed in an alternating manner. First, we perform vanilla ZO optimization as defined in Eq. (2). Second, we identify the ideal projections for the weights and apply them, generating the projected weights. Formally, we define the ideal projection learning as solving the following minimization problem:

$$\min_{\boldsymbol{\gamma}_t} \mathcal{L}(\boldsymbol{\theta}_0 + \frac{\boldsymbol{\gamma}_t}{\|\Delta\boldsymbol{\theta}_t\|}\Delta\boldsymbol{\theta}_t; \mathcal{B}_t) \tag{4}$$

where $\boldsymbol{\gamma}_t = \{\gamma_t^{(\ell)}\}_{\ell=1}^{L}$ is a projection vector at t-th iteration, and L is the number of layers. While searching for the ideal projection, we freeze the model weights and use the same mini-batch data $\mathcal{B}_t$ that is employed for the main ZO weight fine-tuning.

After finding the ideal projection for the t-th ZO step, we project the weights as:

$$\boldsymbol{\theta}_t = \boldsymbol{\theta}_0 + \frac{\boldsymbol{\gamma}_t}{\|\Delta\boldsymbol{\theta}_t\|}\Delta\boldsymbol{\theta}_t \tag{5}$$

where we get the new $\boldsymbol{\theta}_t$ after projection, and then we use the projected one for the following fine-tuning. When the value of $\boldsymbol{\gamma_t}$ is larger than $\|\Delta\boldsymbol{\theta_t}\|$, enlarges the distance gap between the fine-tuned model and the pre-trained model , and vice versa.

3.2. How to Learn the Projection?

Although promising, finding the ideal projection (defined in Eq. (4)) remains challenging due to the high complexity of the objective. A straightforward solution is to also perform backpropagation for gradient computation and optimize the projection accordingly (FO-based method). For example, we use Adam optimizer to directly update $\boldsymbol{\gamma}_t$. The results are shown in Table 2, one can observe that it significantly reduces 67.7% of the iteration and 58.5% of the training GPU hours, and increases by 3.4% in accuracy. These results underscore the effectiveness of incorporating our proposed divergence-driven layer adaptation.

However, searching projection with the FO method makes DiZO only partially gradient-free. Specifically, while the model weights are updated via ZO, the per-layer projection parameter $\gamma_t^{(\ell)}$ is updated via FO, which still requires the backward pass and storing memory-intensive activation. The only memory saving, compared to the vanilla FO fine-tuning, is the optimizer state. As a result, relying on FO to find the ideal projection, though it achieves faster convergence speed and better accuracy in ZO optimization, offers limited overall benefit. It is worth noting that the peak memory usage during training of the FO-based DiZO is similar to that of low-rank adaptation (LoRA) (Hu et al., 2021).

Table 2. Fine-tuning OPT-2.7B on SST-2 dataset. ●: partial gradient-free; DiZO†: learning projection by FO method;

Task Type	Gradient Free	Acc.	#Train Iter.	GPU Hours
MeZO	✓	90.0	100%	100%
DiZO† (w. FO)	●	93.4	33.3%	41.5%
FT	✗	94.2	9.3%	16.8%

Is the projection-based method for enhancing layer-wise divergence in ZO a failed idea that seems promising at first glance but is actually not after deliberation? Fortunately, the answer is no. We develop a ZO projection learning algorithm, which retains the memory-efficient advantages and also achieves training acceleration and accuracy enhancement.

3.3. Projection Learning by Zeroth-order Optimization

Our major goal is to find the ideal projection for adaptive updates while avoiding memory-intensive backpropagation. One potential promising solution is to also utilize the ZO method to update the projection. We estimate the gradient and update the projection as:

$$\nabla\widehat{\mathcal{L}}(\boldsymbol{\gamma}_t; \boldsymbol{\theta}_t) = \left[\frac{\widehat{\mathcal{L}}\left(\boldsymbol{\gamma}_t + \epsilon\boldsymbol{u}; \boldsymbol{\theta}_t\right) - \widehat{\mathcal{L}}\left(\boldsymbol{\gamma}_t - \epsilon\boldsymbol{u}; \boldsymbol{\theta}_t\right)}{2\epsilon}\boldsymbol{u}\right] \tag{6}$$

$$\boldsymbol{\gamma}_{t,j+1} = \boldsymbol{\gamma}_{t,j} - \eta\nabla\widehat{\mathcal{L}}(\boldsymbol{\gamma}_t; \boldsymbol{\theta}_t) \tag{7}$$

where $\boldsymbol{u} \in \mathbb{R}^L$ is a random vector from $\mathcal{N}(0, \mathbf{I})$, $\widehat{\mathcal{L}}$ is the objective defined in Eq. (4).

However, naively applying vanilla ZO optimization for the sub-optimization (projection learning) results in unsatisfactory enhancement. More critically, it can lead to sub-optimization failure and undermine the main fine-tuning process (see Appendix C.2 for results). Two primary issues contribute to the failure. First, the values of projections are not only related to $\boldsymbol{\gamma}_t$ but also the distance gap $\|\Delta\boldsymbol{\theta}_t\|$. Ignoring the distance gap when searching for projections causes uninformative optimization and yields sub-optimal solutions. Second, because the projection is derived through noisy ZO optimization over only a few iterations, there is a risk that the projection magnitude becomes inappropriately small or large. A small projection drives the fine-tuned model too close to the pre-trained model, nullifying many previous updates, while a large projection applies overly ag-

gressive weight updates, destabilizing the training process.

To address the above issues, two strategies are devised.
Re-initialization. To introduce the distance gap $\|\Delta\boldsymbol{\theta}_t\|$ into the projection learning process, the initial value $\gamma_{t,0}$ is reset to $\|\Delta\boldsymbol{\theta}_t\|$ each time the projection is optimized. This means that, initially, the projection magnitude $\frac{\gamma_t}{\|\Delta\boldsymbol{\theta}_t\|} = 1$. If projection updates are not performed, DiZO reverts to standard ZO optimization.
Projection clipping. To prevent drastic weight changes and maintain training stability, we introduce projection clipping. Specifically, given a clipping range $\tau > 0$, if the projection magnitude $\frac{\gamma_t}{\|\Delta\boldsymbol{\theta}_t\|} \notin [1-\tau, 1+\tau]$, it is clipped to remain within this interval. This prevents aggressive model adjustments that could destabilize training.

With the above two strategies, we enhance the learning process of projection, more analysis can be found in Appendix C.2. We also provide a Pytorch-style implementation, please refer to Appendix F for details.

4. Discussion and Overhead Analysis

We have some discussion on our method and analyze the computational overhead here and elaborate further later.

Would adjusting the learning rate be equally effective? As discussed in Section 2.2, our main objective is to provide ZO optimization with diverse-magnitude updates. A seemingly straightforward alternative is to assign different learning rates to each layer. However, in practice, this approach yields results that are similar to or even worse than vanilla ZO in terms of accuracy and GPU hours. We attribute this to the noisy gradient estimation of one single ZO step, which is likely to yield imprecise update directions. Therefore, using unrefined layer-wise learning rates can intensify this noise and further destabilize the optimization process. In contrast, DiZO enables the awareness of the pre-trained model during fine-tuning (see Eq. 5), robustifies the training process (Dong et al., 2021; Oh et al., 2023; Zhai et al., 2023; Wang et al., 2024), and mitigates the noise introduced by ZO's random perturbations. More results and analysis are shown in Appendix C.3.

Memory utilization. Our method requires additional memory as it involves storing the pre-trained model and calculating the weight distance gap with the fine-tuned model, which can become costly when scaling to large LLMs. However, in DiZO, we find that projecting only the weight updates of the *Query* and *Value* layers in the attention module, instead of updating all layers, not only reduces memory requirements but also delivers better performance. As a result, we only need to store the weights of these two types of layers from the pre-trained model, accounting for approximately 16.7% of the parameters in OPT-2.7B, which is a manageable overhead. Similarly, LoRA (Hu et al., 2021) also focuses on weight decomposition for *Query* and *Value* layers, which echoes our observation. Further analysis and results on projection layer selection are provided in Appendix C.1.

Computational overhead. Our method introduces extra computational cost, as the projection is learned alongside the main optimization (fine-tuning). However, we observe that performing projection learning intermittently, only once every few training iterations, does not compromise performance and significantly reduces the added overhead. This strategy reduces the computational burden while maintaining efficiency, allowing DiZO to achieve throughput comparable to vanilla ZO fine-tuning. Additionally, the reduced frequency of projection updates ensures that DiZO remains scalable for larger models and datasets. Please refer to Section 5.4 and Appendix D.1 for more details on computational overhead.

5. Experiments

5.1. Experimental Settings

Models and datasets. We evaluate DiZO with various models, including medium-sized masked models (Liu et al., 2019) (RoBERTa-large) and large-sized autoregressive models (Zhang et al., 2022; Touvron et al., 2023) with different size, including OPT-2.7B, OPT-6.7B, Llama3-3B, and Llama3-8B. The total parameter size is ranging from 355M to 8B. Both classification and generation tasks are included. More details on datasets are shown in Appendix B.1.

Baseline. We mainly compare with two ZO works, memory-efficient ZO optimization (MeZO) (Malladi et al., 2023) and Hessian-informed ZO optimization (HiZOO) (Zhao et al., 2024). MeZO is a fundamental and representative work in ZO LLM fine-tuning but suffers from slow convergence speed. HiZOO[1] is a recently proposed ZO acceleration for LLM fine-tuning, which leverages the estimated second-order information to speed up. In addition, we also incorporate the parameter-efficient fine-tuning (PEFT) technique LoRA (Hu et al., 2021), applying it on top of FO fine-tuning, MeZO, and HiZOO.

Evaluation. For training and evaluation, we follow previous works (Gao et al., 2020; Malladi et al., 2023). We study few-shot and many-shot settings on RoBERTa-large, randomly sampling k samples per class for training and validation, and 1000 samples for testing. For RoBERTa models, we evaluate $k = 16$ and $k = 512$. For OPT and LLaMA, we sample 1000, 500, and 1000 samples for training, validation, and testing. All experiments are conducted on NVIDIA A100 and A6000 GPUs.

[1]We implement HiZOO ourselves, please refer to Appendix B.2 for details.

Table 3. Experiment results on RoBERTa-large (350M) on six classification datasets. Results of the baseline methods MeZO and MeZO LoRA are taken from Malladi et al. (2023). All reported numbers are averaged accuracy with standard deviation shown. Better results between MeZO and DiZO are highlighted in bold.

Dataset	**SST-2**	**SST-5**	**SNLI**	**MNLI**	**RTE**	**TREC**
Task Type	——-sentiment——-		————-language inference————-			–topic–
Zero-shot	79.0	35.5	50.2	48.8	51.4	32.0
Gradient-free methods: $k = 16$						
MeZO	90.5 (1.2)	45.5 (2.0)	66.0 (2.7)	56.5 (2.5)	59.4 (5.3)	76.9 (2.7)
MeZO LoRA	91.4 (0.9)	43.0 (1.6)	69.7 (6.0)	64.0 (2.5)	64.9 (3.6)	73.1 (6.5)
DiZO	**92.2** (0.9)	**47.1** (1.3)	71.0 (3.1)	60.1 (3.5)	**67.9** (4.7)	**77.4** (2.4)
DiZO LoRA	91.7 (0.8)	44.6 (1.7)	**71.6** (3.8)	**65.6** (2.8)	67.3 (3.9)	74.6 (4.3)
Gradient-based methods: $k = 16$						
FT	91.9 (1.8)	47.5 (1.9)	77.5 (2.6)	70.0 (2.3)	66.4 (7.2)	85.0 (2.5)
FT LoRA	91.4 (1.7)	46.7 (1.1)	74.9 (4.3)	67.7 (1.4)	66.1 (3.5)	82.7 (4.1)
Gradient-free methods: $k = 512$						
MeZO	93.3 (0.7)	53.2 (1.4)	83.0 (1.0)	78.3 (0.5)	78.6 (2.0)	94.3 (1.3)
MeZO LoRA	93.4 (0.4)	52.4 (0.8)	84.0 (0.8)	77.9 (0.6)	77.6 (1.3)	95.0 (0.7)
DiZO	**94.6** (0.1)	53.6 (1.7)	**84.5** (0.6)	**79.8** (0.9)	**80.3** (1.8)	93.8 (1.5)
DiZO LoRA	94.3 (0.3)	**54.1** (1.4)	83.7 (1.1)	77.6 (0.4)	79.3 (1.4)	**95.7** (0.9)
Gradient-based methods: $k = 512$						
FT	93.9 (0.7)	55.9 (0.9)	88.7 (0.8)	84.4 (0.8)	82.7 (1.4)	97.3 (0.2)
FT LoRA	94.2 (0.2)	55.3 (0.7)	88.3 (0.5)	83.9 (0.6)	83.2 (1.3)	97.0 (0.3)

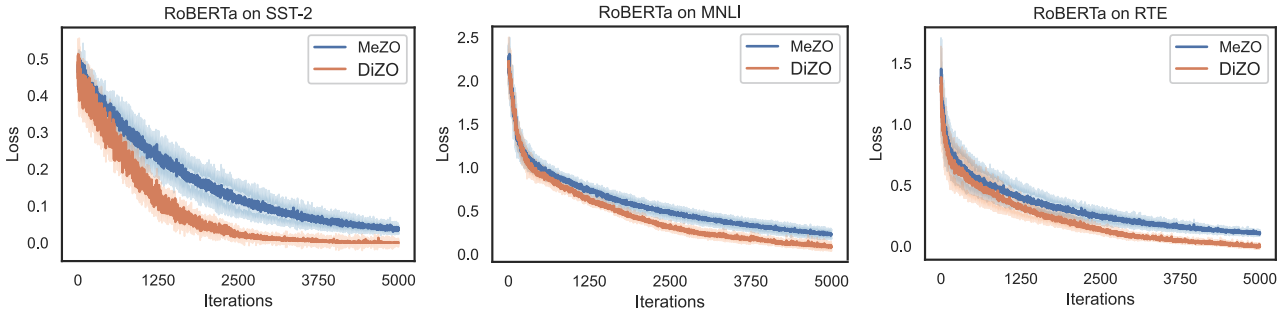

Figure 2. Trajectory of training loss curves when using MeZO and DiZO to fine-tune Roberta-large on SST-2, MNLI, and RTE.

5.2. Medium-sized masked language models

We conduct experiments on RoBERTa-large across three types of datasets and compare DiZO with two ZO baselines. We also explore PEFT by integrating LoRA. Table 3 presents the results, while Figure 2 shows the trajectory of training loss curves, indicating the convergence speed of DiZO and MeZO. Our key findings are as follows:

DiZO greatly increases the convergence speed over MeZO. By using divergence-driven layer adaptation, the loss curve of DiZO decreases much faster, cutting the required iterations by over 50% on SST-2, MNLI, and RTE. In addition, DiZO improves accuracy by 1.7%, 3.6%, and 8.5% on these three datasets, respectively.

DiZO outperforms MeZO and achieves results on par with full fine-tuning. From Table 3, DiZO consistently surpasses MeZO on all six datasets. Notably, on SST-2 and RTE datasets, DiZO even shows better performance than FO full-parameter fine-tuning, increasing by 0.3% and 1.5%, respectively.

DiZO is effective for both full-parameter fine-tuning and PEFT. Although DiZO applies projections based on the distance with the pre-trained model, while such prior knowledge does not exist for the decomposed weights of LoRA, it still delivers some gains.

5.3. Large autoregressive language models

To assess the broader applicability of DiZO, we run experiments on the OPT and Llama series autoregressive LLMs

Table 4. Experiments results of fine-tuning OPT-2.7B on seven classification datasets and two text generation datasets (with 1000 training samples). Better results between MeZO, HiZOO, and DiZO are highlighted in bold.

Dataset	**SST-2**	**RTE**	**CB**	**BoolQ**	**WSC**	**WIC**	**MultiRC**	**SQuAD**	**DROP**
Task Type	———	———	———	classification	———	———	———	——generation——	
Zero-shot	56.3	54.2	50.0	47.6	36.5	52.7	44.4	29.8	10.0
FT	94.2	81.2	82.1	72.2	63.8	65.8	71.6	78.4	30.3
LoRA	94.6	80.8	82.7	77.7	59.8	64.0	72.8	77.9	31.1
MeZO	90.0	63.5	69.6	67.4	61.5	57.6	**58.7**	68.7	22.9
HiZOO	90.8	60.6	70.4	**68.0**	60.2	56.6	54.8	68.3	23.4
DiZO	**92.5**	**68.2**	**71.4**	67.0	**63.4**	**57.9**	56.4	**69.0**	**24.3**
MeZO LoRA	91.4	66.6	71.1	67.6	59.6	57.0	57.0	70.8	22.5
HiZOO LoRA	90.6	65.2	71.4	67.4	52.6	**58.8**	**59.0**	71.8	22.7
DiZO LoRA	**91.5**	**68.4**	**71.8**	**70.0**	**61.6**	58.4	56.2	**74.4**	**23.3**

Table 5. Experiment results on OPT-6.7B for four classification datasets and one text generation dataset (with 1000 training samples). Better results are highlighted in bold.

Dataset	**SST-2**	**RTE**	**CB**	**WSC**	**SQuAD**
Task Type	———	classification	———		–generation–
MeZO	90.2	73.2	71.4	**62.2**	76.0
HiZOO	90.7	74.2	71.8	62.1	77.3
DiZO	**91.1**	**74.8**	**73.2**	61.8	**78.6**
MeZO LoRA	91.6	71.2	71.4	61.8	76.3
HiZOO LoRA	91.3	**71.3**	71.4	62.1	76.1
DiZO LoRA	**92.4**	70.2	**71.8**	**62.6**	**77.9**

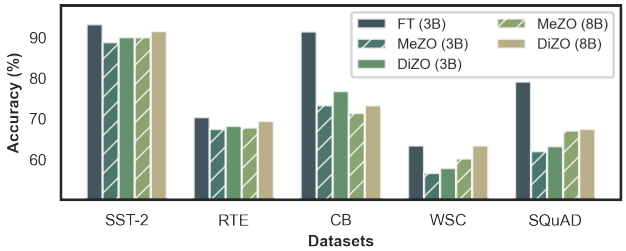

Figure 3. Experiment result on Llama3-3B and Llama3-8B for four classification datasets and one text generation dataset. More results and detailed numbers are shown in Appendix D.2.

covering both classification and generation tasks. The overall results are summarized in Table 4, Table 5, and Figure 3 for OPT-2.7B, OPT-6.7B, and Llama series, respectively. We also compare the convergence speeds of DiZO and MeZO on OPT-2.7B across multiple datasets in Figure 4. Below, we highlight the key observations from these experiments.

DiZO dramatically reduces the training GPU hours compared with the representative baseline MeZO. By incorporating divergence-driven layer adaptation, DiZO quickly establishes meaningful divergence across layers, whereas MeZO requires many more iterations to achieve the desired layer-wise divergence. As shown in Table 4, DiZO converges with far fewer iterations across nine datasets, resulting in up to a 48% reduction in training GPU hours. Moreover, unlike HiZOO, which reduces the number of iterations needed but slows the throughput of MeZO by more than 1.5× due to Hessian estimation, DiZO keeps its throughput nearly on par with MeZO. This efficiency is achieved because the additional projection learning procedure needs only two forward passes and is performed intermittently.

DiZO outperforms baselines in both standard and parameter-efficient settings. From Table 4, DiZO surpasses MeZO and HiZOO with or without the LoRA, achieving results comparable to FO methods. Across seven classification datasets, DiZO ranks first on five, and it also leads in both text generation tasks. Table 5 shows that these advantages persist even when scaling up to OPT-6.7B. Moreover, as illustrated in Figure 3, the fine-tuning process of Llama-series model also benefits from layer-wise adaptive updates.

5.4. Memory and Speed Analysis

In this section, we examine the memory utilization and convergence speed of DiZO in comparison with both ZO baselines and FO fine-tuning approaches (with and without LoRA). Table 6 presents the results of fine-tuning OPT-2.7B on the RTE dataset, more results are shown in Appendix D.1.

From the memory perspective, DiZO maintains the advantage of avoiding backpropagation, getting rid of the storage of memory-intensive data, and reducing memory usage by about 90% compared to FO fine-tuning. As explained in Section 4, the additional memory requirement of DiZO stems from storing a portion of the pre-trained weights, including the weight of the *Query* and *Value*, amounting to only 16.7% of the total parameters. In contrast, HiZOO needs to store Hessian information for all layers, with memory usage proportional to the size of the parameters.

From the perspective of convergence speed, DiZO greatly reduces the required iterations while maintaining throughput

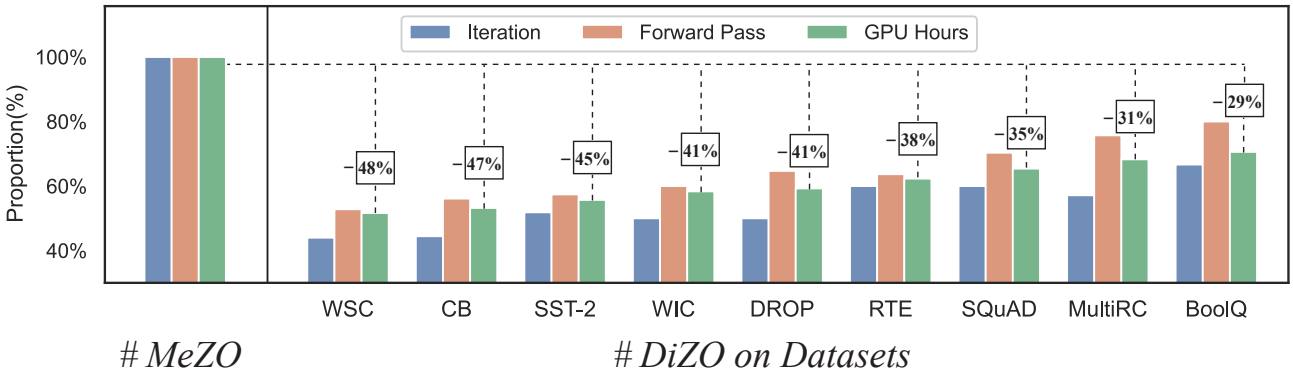

Figure 4. Comparison of convergence iterations, forward pass, and training GPU hours between MeZO and DiZO across multiple datasets. Results are presented as proportions, with the percentage of saved GPU hours highlighted for each dataset.

Table 6. Memory utilization and speed test on OPT-2.7B on RTE dataset (180 tokens per example on average). ●: partial gradient-free; ✓: gradient-free; ✗: gradient-based. DiZO†: learning projection with Adam. For a fair comparison, the speed and memory are measured on the same machine with the same setting using the same batch size. Please refer to Appendix D.1 for results on more datasets.

Task Type	**Gradient Free**	**LoRA Added**	**Peak Memory**	**Averaged Memory**	**Throughput**	**#Train Iter.**	**GPU Hours**
FT	✗	✗	62.2 GB	62.2 GB	1.05 it/s	10.0%	16.2%
LoRA	✗	✓	42.5 GB	42.5 GB	2.12 it/s	8.3%	6.6%
DiZO†	●	✗	44.7 GB	10.1 GB	1.43 it/s	33.3%	39.6%
DiZO LoRA†	●	✓	40.1 GB	9.8 GB	2.40 it/s	26.6%	18.8%
MeZO	✓	✗	7.8 GB	7.8 GB	1.70 it/s	100.0%	100.0%
HiZOO	✓	✗	13.2 GB	13.2 GB	1.21 it/s	63.3%	88.9%
DiZO	✓	✗	9.5 GB	9.5 GB	1.54 it/s	60.0%	62.3%
MeZO LoRA	✓	✓	7.7 GB	7.7 GB	3.10 it/s	94.2%	51.6%
HiZOO LoRA	✓	✓	13.0 GB	13.0 GB	2.07 it/s	80.0%	65.7%
DiZO LoRA	✓	✓	9.4 GB	9.4 GB	2.87 it/s	66.7%	39.5%

similar to MeZO, resulting in significantly fewer training GPU hours. In contrast, HiZOO does not achieve comparable iteration savings and lowers the throughput of MeZO by about 1.5× because it requires Hessian information estimation. As a result, it only shows a modest acceleration in training GPU hours, in some settings, such as HiZOO with LoRA on RTE, it even consumes more training GPU hours than MeZO with LoRA.

A notable byproduct of our method is using a FO approach (e.g., with the Adam optimizer) to learn the projections. While this version has memory consumption comparable to LoRA and requires additional training GPU hours, it offers distinct advantages. Since DiZO does not update projections at every iteration, FO-based DiZO exhibits significantly lower average memory usage than FO-based LoRA, with an average memory overhead close to that of the ZO-based DiZO. Although average memory usage may seem less critical in single-process, single-GPU setup, many real-world on-device training scenarios involve multi-process environments (Li et al., 2024; Ye et al., 2024). In such cases, the FO-based DiZO can stagger memory usage phases across processes, enabling parallel operations that purely FO methods cannot achieve. Furthermore, compared with ZO-based DiZO, the FO version reduces extra training GPU hours and delivers better performance. These qualities make it particularly appealing for specific on-device training cases.

6. Conclusion

In this paper, we propose a novel layer-wise divergence analysis to reveal the distinct update pattern between FO and ZO methods. Building on these insights, we present DiZO, an enhanced ZO method using divergence-driven layer adaptation to resemble the learning capacity of the FO method. DiZO achieves significant training acceleration and superior performance across diverse tasks and architectures. Moreover, our method can be seamlessly integrated with PEFT techniques like LoRA for additional speedup. For future work, we plan to explore DiZO in other fields, particularly for fine-tuning large pre-trained vision models.

7. Impact Statement

This paper presents work whose goal is to advance the field of Machine Learning. There are many potential societal consequences of our work, none of which we feel must be specifically highlighted here.

References

Ainslie, J., Lee-Thorp, J., de Jong, M., Zemlyanskiy, Y., Lebrón, F., and Sanghai, S. Gqa: Training generalized multi-query transformer models from multi-head checkpoints. *arXiv preprint arXiv:2305.13245*, 2023.

Bar-Haim, R., Dagan, I., Dolan, B., Ferro, L., Giampiccolo, D., Magnini, B., and Szpektor, I. The second pascal recognising textual entailment challenge. In *Proceedings of the second PASCAL challenges workshop on recognising textual entailment*, volume 1. Citeseer, 2006.

Bentivogli, L., Clark, P., Dagan, I., and Giampiccolo, D. The fifth pascal recognizing textual entailment challenge. *TAC*, 7(8):1, 2009.

Bowman, S. R., Angeli, G., Potts, C., and Manning, C. D. A large annotated corpus for learning natural language inference. *arXiv preprint arXiv:1508.05326*, 2015.

Chen, A., Zhang, Y., Jia, J., Diffenderfer, J., Liu, J., Parasyris, K., Zhang, Y., Zhang, Z., Kailkhura, B., and Liu, S. Deepzero: Scaling up zeroth-order optimization for deep model training. *arXiv preprint arXiv:2310.02025*, 2023.

Chen, H., Zhang, J., Du, Y., Xiang, S., Yue, Z., Zhang, N., Cai, Y., and Zhang, Z. Understanding the potential of fpga-based spatial acceleration for large language model inference. *ACM Transactions on Reconfigurable Technology and Systems*, 2024.

Chen, J., Guo, H., Yi, K., Li, B., and Elhoseiny, M. Visualgpt: Data-efficient adaptation of pretrained language models for image captioning. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pp. 18030–18040, 2022.

Chen, P.-Y., Zhang, H., Sharma, Y., Yi, J., and Hsieh, C.-J. Zoo: Zeroth order optimization based black-box attacks to deep neural networks without training substitute models. In *Proceedings of the 10th ACM workshop on artificial intelligence and security*, pp. 15–26, 2017.

Chowdhery, A., Narang, S., Devlin, J., Bosma, M., Mishra, G., Roberts, A., Barham, P., Chung, H. W., Sutton, C., Gehrmann, S., et al. Palm: Scaling language modeling with pathways. *Journal of Machine Learning Research*, 24(240):1–113, 2023.

Clark, C., Lee, K., Chang, M.-W., Kwiatkowski, T., Collins, M., and Toutanova, K. Boolq: Exploring the surprising difficulty of natural yes/no questions. *arXiv preprint arXiv:1905.10044*, 2019.

Dagan, I., Glickman, O., and Magnini, B. The pascal recognising textual entailment challenge. In *Machine learning challenges workshop*, pp. 177–190. Springer, 2005.

De Marneffe, M.-C., Simons, M., and Tonhauser, J. The commitmentbank: Investigating projection in naturally occurring discourse. In *proceedings of Sinn und Bedeutung*, volume 23, pp. 107–124, 2019.

Devlin, J. Bert: Pre-training of deep bidirectional transformers for language understanding. *arXiv preprint arXiv:1810.04805*, 2018.

Dhurandhar, A., Chen, P.-Y., Luss, R., Tu, C.-C., Ting, P., Shanmugam, K., and Das, P. Explanations based on the missing: Towards contrastive explanations with pertinent negatives. *Advances in neural information processing systems*, 31, 2018.

Dhurandhar, A., Pedapati, T., Balakrishnan, A., Chen, P.-Y., Shanmugam, K., and Puri, R. Model agnostic contrastive explanations for structured data. *arXiv preprint arXiv:1906.00117*, 2019.

Dong, X., Luu, A. T., Lin, M., Yan, S., and Zhang, H. How should pre-trained language models be fine-tuned towards adversarial robustness? *Advances in Neural Information Processing Systems*, 34:4356–4369, 2021.

Dua, D., Wang, Y., Dasigi, P., Stanovsky, G., Singh, S., and Gardner, M. Drop: A reading comprehension benchmark requiring discrete reasoning over paragraphs. *arXiv preprint arXiv:1903.00161*, 2019.

Gao, T., Fisch, A., and Chen, D. Making pre-trained language models better few-shot learners. *arXiv preprint arXiv:2012.15723*, 2020.

Gautam, T., Park, Y., Zhou, H., Raman, P., and Ha, W. Variance-reduced zeroth-order methods for fine-tuning language models. *arXiv preprint arXiv:2404.08080*, 2024.

Gholami, A., Yao, Z., Kim, S., Hooper, C., Mahoney, M. W., and Keutzer, K. Ai and memory wall. *IEEE Micro*, 2024.

Giampiccolo, D., Magnini, B., Dagan, I., and Dolan, W. B. The third pascal recognizing textual entailment challenge. In *Proceedings of the ACL-PASCAL workshop on textual entailment and paraphrasing*, pp. 1–9, 2007.

Gu, J., Feng, C., Zhao, Z., Ying, Z., Chen, R. T., and Pan, D. Z. Efficient on-chip learning for optical neural networks through power-aware sparse zeroth-order optimization. In *Proceedings of the AAAI conference on artificial intelligence*, volume 35, pp. 7583–7591, 2021.

Gururangan, S., Marasović, A., Swayamdipta, S., Lo, K., Beltagy, I., Downey, D., and Smith, N. A. Don't stop pretraining: Adapt language models to domains and tasks. *arXiv preprint arXiv:2004.10964*, 2020.

Hoffmann, J., Borgeaud, S., Mensch, A., Buchatskaya, E., Cai, T., Rutherford, E., de Las Casas, D., Hendricks, L. A., Welbl, J., Clark, A., et al. An empirical analysis of compute-optimal large language model training. *Advances in Neural Information Processing Systems*, 35: 30016–30030, 2022.

Hu, E. J., Shen, Y., Wallis, P., Allen-Zhu, Z., Li, Y., Wang, S., Wang, L., and Chen, W. Lora: Low-rank adaptation of large language models. *arXiv preprint arXiv:2106.09685*, 2021.

Hur, S., Na, S., Kwon, D., Kim, J., Boutros, A., Nurvitadhi, E., and Kim, J. A fast and flexible fpga-based accelerator for natural language processing neural networks. *ACM Transactions on Architecture and Code Optimization*, 20 (1):1–24, 2023.

Johnson, R. and Zhang, T. Accelerating stochastic gradient descent using predictive variance reduction. *Advances in neural information processing systems*, 26, 2013.

Kaplan, J., McCandlish, S., Henighan, T., Brown, T. B., Chess, B., Child, R., Gray, S., Radford, A., Wu, J., and Amodei, D. Scaling laws for neural language models. *arXiv preprint arXiv:2001.08361*, 2020.

Khashabi, D., Chaturvedi, S., Roth, M., Upadhyay, S., and Roth, D. 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)*, pp. 252–262, 2018.

Levesque, H., Davis, E., and Morgenstern, L. The winograd schema challenge. In *Thirteenth international conference on the principles of knowledge representation and reasoning*, 2012.

Li, X., Li, Y., Li, Y., Cao, T., and Liu, Y. Flexnn: Efficient and adaptive dnn inference on memory-constrained edge devices. In *Proceedings of the 30th Annual International Conference on Mobile Computing and Networking*, pp. 709–723, 2024.

Li, X. L. and Liang, P. Prefix-tuning: Optimizing continuous prompts for generation. *arXiv preprint arXiv:2101.00190*, 2021.

Liu, S., Kailkhura, B., Chen, P.-Y., Ting, P., Chang, S., and Amini, L. Zeroth-order stochastic variance reduction for nonconvex optimization. *Advances in Neural Information Processing Systems*, 31, 2018.

Liu, Y., Ott, M., Goyal, N., Du, J., Joshi, M., Chen, D., Levy, O., Lewis, M., Zettlemoyer, L., and Stoyanov, V. Roberta: A robustly optimized BERT pretraining approach. *CoRR*, abs/1907.11692, 2019. URL http://arxiv.org/abs/1907.11692.

Liu, Y., Zhu, Z., Gong, C., Cheng, M., Hsieh, C.-J., and You, Y. Sparse mezo: Less parameters for better performance in zeroth-order llm fine-tuning. *arXiv preprint arXiv:2402.15751*, 2024.

Malladi, S., Gao, T., Nichani, E., Damian, A., Lee, J. D., Chen, D., and Arora, S. Fine-tuning language models with just forward passes. *Advances in Neural Information Processing Systems*, 36:53038–53075, 2023.

Oh, C., Kim, M., Lim, H., Park, J., Jeong, E., Cheng, Z.-Q., and Song, K. Towards calibrated robust fine-tuning of vision-language models. *arXiv preprint arXiv:2311.01723*, 2023.

Ouyang, L., Wu, J., Jiang, X., Almeida, D., Wainwright, C., Mishkin, P., Zhang, C., Agarwal, S., Slama, K., Ray, A., et al. Training language models to follow instructions with human feedback. *Advances in neural information processing systems*, 35:27730–27744, 2022.

Pilehvar, M. T. and Camacho-Collados, J. Wic: the word-in-context dataset for evaluating context-sensitive meaning representations. *arXiv preprint arXiv:1808.09121*, 2018.

Radford, A., Kim, J. W., Hallacy, C., Ramesh, A., Goh, G., Agarwal, S., Sastry, G., Askell, A., Mishkin, P., Clark, J., et al. Learning transferable visual models from natural language supervision. In *International conference on machine learning*, pp. 8748–8763. PMLR, 2021.

Rajpurkar, P. Squad: 100,000+ questions for machine comprehension of text. *arXiv preprint arXiv:1606.05250*, 2016.

Sener, O. and Koltun, V. Learning to guide random search. *arXiv preprint arXiv:2004.12214*, 2020.

Shamir, O. An optimal algorithm for bandit and zero-order convex optimization with two-point feedback. *Journal of Machine Learning Research*, 18(52):1–11, 2017.

Shu, Y., Dai, Z., Sng, W., Verma, A., Jaillet, P., and Low, B. K. H. Zeroth-order optimization with trajectory-informed derivative estimation. In *The Eleventh International Conference on Learning Representations*, 2023.

Singh, M., Gustafson, L., Adcock, A., de Freitas Reis, V., Gedik, B., Kosaraju, R. P., Mahajan, D., Girshick, R., Dollár, P., and Van Der Maaten, L. Revisiting weakly supervised pre-training of visual perception models. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pp. 804–814, 2022.

Socher, R., Perelygin, A., Wu, J., Chuang, J., Manning, C. D., Ng, A. Y., and Potts, C. Recursive deep models for semantic compositionality over a sentiment treebank. In *Proceedings of the 2013 conference on empirical methods in natural language processing*, pp. 1631–1642, 2013.

Talmor, A., Herzig, J., Lourie, N., and Berant, J. Commonsenseqa: A question answering challenge targeting commonsense knowledge. *arXiv preprint arXiv:1811.00937*, 2018.

Touvron, H., Lavril, T., Izacard, G., Martinet, X., Lachaux, M.-A., Lacroix, T., Rozière, B., Goyal, N., Hambro, E., Azhar, F., et al. Llama: Open and efficient foundation language models. *arXiv preprint arXiv:2302.13971*, 2023.

Vemula, A., Sun, W., and Bagnell, J. Contrasting exploration in parameter and action space: A zeroth-order optimization perspective. In *The 22nd International Conference on Artificial Intelligence and Statistics*, pp. 2926–2935. PMLR, 2019.

Verma, A., Bangar, S., Subramanyam, A. V., Lal, N., Shah, R. R., and Satoh, S. Certified zeroth-order black-box defense with robust unet denoiser. *arXiv preprint arXiv:2304.06430*, 2023.

Voorhees, E. M. and Tice, D. M. Building a question answering test collection. In *Proceedings of the 23rd annual international ACM SIGIR conference on Research and development in information retrieval*, pp. 200–207, 2000.

Wang, A., Pruksachatkun, Y., Nangia, N., Singh, A., Michael, J., Hill, F., Levy, O., and Bowman, S. Superglue: A stickier benchmark for general-purpose language understanding systems. *Advances in neural information processing systems*, 32, 2019.

Wang, S., Zhang, J., Yuan, Z., and Shan, S. Pre-trained model guided fine-tuning for zero-shot adversarial robustness. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pp. 24502–24511, 2024.

Wang, X., Guo, W., Su, J., Yang, X., and Yan, J. Zarts: On zero-order optimization for neural architecture search. *Advances in Neural Information Processing Systems*, 35: 12868–12880, 2022.

Yang, W., Xie, Y., Lin, A., Li, X., Tan, L., Xiong, K., Li, M., and Lin, J. End-to-end open-domain question answering with bertserini. *arXiv preprint arXiv:1902.01718*, 2019.

Yao, Z., Gholami, A., Keutzer, K., and Mahoney, M. W. Pyhessian: Neural networks through the lens of the hessian. In *2020 IEEE international conference on big data (Big data)*, pp. 581–590. IEEE, 2020.

Ye, H., Huang, Z., Fang, C., Li, C. J., and Zhang, T. Hessian-aware zeroth-order optimization for black-box adversarial attack. *arXiv preprint arXiv:1812.11377*, 2018.

Ye, S., Zeng, L., Chu, X., Xing, G., and Chen, X. Asteroid: Resource-efficient hybrid pipeline parallelism for collaborative dnn training on heterogeneous edge devices. In *Proceedings of the 30th Annual International Conference on Mobile Computing and Networking*, pp. 312–326, 2024.

Zeng, S., Liu, J., Dai, G., Yang, X., Fu, T., Wang, H., Ma, W., Sun, H., Li, S., Huang, Z., et al. Flightllm: Efficient large language model inference with a complete mapping flow on fpgas. In *Proceedings of the 2024 ACM/SIGDA International Symposium on Field Programmable Gate Arrays*, pp. 223–234, 2024.

Zhai, Y., Tong, S., Li, X., Cai, M., Qu, Q., Lee, Y. J., and Ma, Y. Investigating the catastrophic forgetting in multimodal large language models. *arXiv preprint arXiv:2309.10313*, 2023.

Zhang, S., Roller, S., Goyal, N., Artetxe, M., Chen, M., Chen, S., Dewan, C., Diab, M., Li, X., Lin, X. V., et al. Opt: Open pre-trained transformer language models. *arXiv preprint arXiv:2205.01068*, 2022.

Zhang, Y., Li, P., Hong, J., Li, J., Zhang, Y., Zheng, W., Chen, P.-Y., Lee, J. D., Yin, W., Hong, M., et al. Revisiting zeroth-order optimization for memory-efficient llm fine-tuning: A benchmark. *arXiv preprint arXiv:2402.11592*, 2024.

Zhao, Y., Dang, S., Ye, H., Dai, G., Qian, Y., and Tsang, I. W. Second-order fine-tuning without pain for llms: A hessian informed zeroth-order optimizer. *arXiv preprint arXiv:2402.15173*, 2024.

Zheng, M., Gao, P., Zhang, R., Li, K., Wang, X., Li, H., and Dong, H. End-to-end object detection with adaptive clustering transformer. *arXiv preprint arXiv:2011.09315*, 2020.

A. Related Work

A.1. Fine-tuning of Pre-trained Models

Fine-tuning a pre-trained model offers a powerful way to reuse learned representations and reduce training costs compared to building models from scratch, often achieving superior performance (Gururangan et al., 2020; Ouyang et al., 2022). Initially successful in NLP with models like BERT, RoBERTa, and GPT (Devlin, 2018; Liu et al., 2019; Chen et al., 2022), fine-tuning has also shown promise in vision tasks such as CLIP and SWAG (Radford et al., 2021; Singh et al., 2022). Recent parameter-efficient fine-tuning (PEFT), including LoRA (Hu et al., 2021), and prefix tuning (Li & Liang, 2021), further minimize resource needs by updating only a small subset of parameters, preserving most of the pre-trained weights and ensuring valuable knowledge is retained.

A.2. Zeroth-order Optimization and Acceleration

ZO optimization emerges as an attractive technique that optimizes the model without backpropagation (Chen et al., 2023; 2017; Ye et al., 2018; Verma et al., 2023; Dhurandhar et al., 2018; 2019). Unlike most frequently used FO optimization which directly obtains and leverages the gradient for optimization, the zeroth-order method utilizes objective function value oracle only, estimating the gradient by finite differences. ZO method has a wide range of applications in machine learning fields, including adversarial attack and defense (Chen et al., 2017; Ye et al., 2018; Verma et al., 2023), machine learning explainability (Dhurandhar et al., 2018; 2019), reinforcement learning (Vemula et al., 2019), and on-chip training (Gu et al., 2021). Recently, the ZO method has been proposed to be leveraged on LLM fine-tuning to address the significant memory usage. Malladi et al. (2023) proposed MeZO, first scaling ZO optimization to fine-tuning parameter-intensive LLMs, greatly reducing memory utilization. On top of MeZO, Zhao et al. (2024) proposed HiZOO, leveraging the estimated Hessian information for better learning capacity, but reducing the throughput of MeZO to some extent.

ZO optimization, although it significantly saves memory, converges more slowly than FO methods due to higher variance from random search. Liu et al. (2018) introduced ZO-SVRG by incorporating variance reduction techniques (Johnson & Zhang, 2013). Shu et al. (2023) proposed using a Gaussian process to model objective function queries, thereby reducing query complexity and allowing more frequent queries to lower gradient variance. Sener & Koltun (2020) performed random search on a learned low-dimensional manifold, reducing the number of needed objective queries. However, existing ZO accelerators face two main challenges when adapting to ZO fine-tuning for LLMs. First, these approaches were typically designed for smaller-scale tasks involving fewer parameters and less data, and cannot be directly extended to large-scale LLMs. Second, many prior methods focus on improving query efficiency, whereas recent work has shown that a single query can suffice for LLM fine-tuning (Malladi et al., 2023). How to effectively accelerate ZO optimization on large model fine-tuning remains a problem.

B. Experiment Settings and Analysis

B.1. Datasets and Evaluation

For the RoBERTa-large model, we use the following classification datasets: SST-2 (Socher et al., 2013), SST-5 (Socher et al., 2013), SNLI (Bowman et al., 2015), TREC (Voorhees & Tice, 2000), MNLI (Yao et al., 2020), and RTE (Dagan et al., 2005; Bar-Haim et al., 2006; Bentivogli et al., 2009; Giampiccolo et al., 2007). Following previous studies, we cap the test set size at 1000 samples. Two training settings are considered: $k = 16$ and $k = 512$, where we randomly select 16 or 512 samples per class for both training and validation.

For the OPT and Llama series models, we use the SuperGLUE benchmark (Wang et al., 2019), which includes RTE (Dagan et al., 2005; Bar-Haim et al., 2006; Bentivogli et al., 2009; Giampiccolo et al., 2007), CB (De Marneffe et al., 2019), BoolQ (Clark et al., 2019), WIC (Pilehvar & Camacho-Collados, 2018), WSC (Levesque et al., 2012), and MultiRC (Khashabi et al., 2018). We also include SST-2 (Socher et al., 2013) and two question answering datasets, SQuAD (Rajpurkar, 2016) and DROP (Dua et al., 2019). For each of these datasets, we randomly sample 1000 instances for training, 500 for validation, and 1000 for testing.

B.2. Implementation of Baselines

Memory-efficient ZO (MeZO) MeZO (Malladi et al., 2023) serves as a fundamental baseline for fine-tuning large language models (LLMs) using zeroth-order (ZO) optimization. By resampling perturbations with a fixed random seed, MeZO

Table 7. The hyperparameter for experiments. For DiZO and DiZO LoRA, we only show the setting of extra hyperparameters, and have the same setting in other common hyperparameters with MeZO and MeZO LoRA respectively.

Experiment	Hyperparameters	Values
FT	Batch size	8
	Learning rate	{1e-5, 5e-5}
	Lr schedule	Constant for RoBERTa Linear for OPT and Llama
MeZO	Batch size	{64, 16}
	Learning rate η (Lr)	{1e-6, 5e-7}
	ϵ	1e-3
	Lr schedule	Constant for RoBERTa Linear for OPT and Llama
MeZO LoRA	Batch size	{64, 16}
	Learning rate η (Lr)	{1e-4, 5e-5}
	ϵ	1e-2
	Lr schedule	Constant for RoBERTa Linear for OPT and Llama
DiZO (LoRA)	Projection update cycle	{50, 100, 200, 400}
	Smoothing scalar $\epsilon^{'}$	{1e-1, 5e-2}
	Clip range τ	{0.1, 0.2, 0.3}

eliminates the need to store perturbations that are the same size as the model, thereby saving memory. For our implementation of MeZO, we adapted the code released by the authors at `https://github.com/princeton-nlp/MeZO` with minimal modifications.

Hessian-informed ZO (HiZOO) HiZOO (Zhao et al., 2024) is a recently proposed method for ZO fine-tuning of LLMs that leverages estimated second-order information to accelerate optimization. During the implementation of HiZOO, we identified several bugs in the released code at `https://anonymous.4open.science/r/HiZOO-27F8`, such as overflows when computing the Hessian. Consequently, we implemented the baseline ourselves. Additionally, we used the parameter settings from the original code instead of those described in the paper, as they resulted in better performance according to our implementation.

B.3. Hyperparameter Setting

We use the hyperparameters in Table 7 for experiments on RoBERTa-large, OPT-series, and Llama-series models. Specifically, the choice of clip range did not significantly impact the performance. The selection of the projection update cycle and scalar for projection affects the performance somewhat. Generally, for datasets that need larger iterations for convergence, or for these harder datasets, DiZO prefers a larger update cycle, while for those less complicated datasets, DiZO benefits from a smaller update cycle.

C. Closer look at DiZO

C.1. Ablation for Projection Layers Selection

Instead of applying projections to all layers, which would require storing the entire pre-trained model, we focus only on projecting the weights of the *Query* and *Value* in the attention modules. As shown in Table 8, this strategy achieves the best trade-off between the overall performance and extra storage requirements, does not reduce the performance and only 16.7% of the parameters of the pre-trained model are needed to store. A Similar strategy has also been adopted in LoRA (Hu et al., 2021).

C.2. Ablation for Strategies in ZO Projection Learning

As discussed in Section 3.3, we introduce two strategies, *re-initialization* (Re-init) and *projection clipping* (Clipping), to enhance projection learning and improve the stability of fine-tuning. The ablation results for these strategies, along with the

Table 8. Ablation study for selecting which layers to project. The highlighted line with a blue rectangle is the setting used in DiZO. Extra memory indicates the extra memory needed due to pre-trained model storing. Attn_Q: attention Query layer; Attn_V: attention Value layer; Attn_K: attention Key layer; Attn_O: attention output projection; Dense: dense fully connected layer.

Attn_Q	Attn_V	Attn_K	Attn_O	Dense	Extra memory	SST-2	RTE	SQuAD
✓	✓	✓	✓	✓	100%	91.7	68.4	67.3
✓	✓	✓	✓	✗	33.3%	92.2	67.9	69.2
✓	✓	✓	✗	✗	25.0%	91.9	67.1	68.1
✓	✓	✗	✗	✗	16.7%	92.5	68.2	69.0
✓	✗	✗	✗	✗	8.4%	90.5	64.9	66.5

corresponding loss curves, are shown in Figure 5.

Overall (left in Figure 5), omitting either Re-init or Clipping significantly diminishes the benefits of DiZO, with MeZO outperforming DiZO in these cases. Specifically, without Re-init, accuracy drops sharply, falling below MeZO. Similarly, without Clipping, while DiZO slightly outperforms MeZO on simpler datasets like SST-2, it suffers from severe model collapse on more challenging datasets, leading to a significant decline in accuracy.

From the loss curve trajectory (right in Figure 5), without Re-init, DiZO loses its advantage in training acceleration, as the loss curve becomes noticeably slower to decrease. Without Clipping, the loss curve exhibits significant oscillations during certain training steps. This instability arises when projections are optimized to unsuitable values, such as extremely large or small magnitudes. These inappropriate projections cause substantial changes in model weights, leading to pronounced oscillations in the loss.

Type	Re-init	Clipping	SST-2	SNLI	TREC
MeZO	-	-	90.5	66.0	76.9
DiZO	✗	✓	88.6	64.2	73.8
	✓	✗	90.9	56.2	61.2
	✓	✓	92.2	71.6	77.4

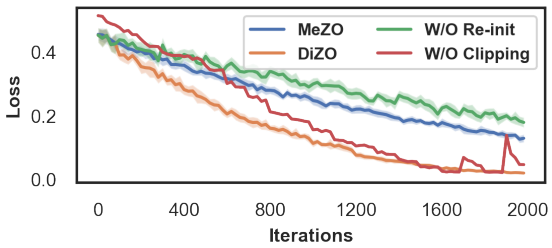

Figure 5. Ablation study for the two strategies: re-initialization and projection clipping, which is conducted on RoBERTa-large ($k = 16$). Left: overall results when ablating the strategies. Right: loss curve when ablating the strategies.

C.3. Does Other Alternative Strategies for Layer-wise Divergence Work?

As discussed in Section 2.2, our objective is to enhance layer-wise divergence in ZO optimization. Naturally, with consideration of this objective, one may raise two questions regarding the projection strategy we adopt: 1) Can we perform layer-wise projections on the learning rate? 2) When updating weight by projection at t-th iteration, why do we use the weights of pre-trained model $\boldsymbol{\theta}_0$ as the base of the update (shown in Eq.5) instead of the weights from the $(t-1)$-th iteration, $\boldsymbol{\theta}_{t-1}$?

Table 9. Comparison on conducting projection on learning rate (LR) or use weight at $(t-1)$-th iteration $\boldsymbol{\theta}_{t-1}$ instead of the weight of the pre-trained model $\boldsymbol{\theta}_0$ as the base of projection. Results are obtained by fine-tuning OPT-2.7B.

Dataset	SST-2		RTE		SQuAD	
	Acc.	GPU Hours	Acc.	GPU Hours	F1.	GPU Hours
MeZO	90.0	100.0%	63.5	100.0%	68.7	100.0%
LR projection	89.5	94.7%	63.9	108.5%	67.9	89.8%
$\boldsymbol{\theta}_{t-1}$ projection	90.7	87.8%	64.5	90.3%	67.2	88.4%
DiZO	92.5	55.7%	68.2	62.3%	69.0	65.4%

Table 10. Memory utilization and speed test on OPT-2.7B on SST-2 dataset (35 tokens per example on average). 🔵: partial gradient-free; ✓: gradient-free; ✗: gradient-based. DiZO†: searching projection with Adam.

Task Type	Gradient Free	LoRA Added	Peak Memory	Averaged Memory	Throughput	#Train Iter.	GPU Hours
FT	✗	✗	45.4 GB	45.4 GB	1.81 it/s	9.3%	16.8%
LoRA	✗	✓	18.4 GB	18.4 GB	4.50 it/s	5.6%	4.3%
DiZO†	🔵	✗	17.8 GB	15.7 GB	2.63 it/s	33.3%	41.5%
DiZO LoRA†	🔵	✓	16.1 GB	14.7 GB	4.16 it/s	22.2%	17.5%
MeZO	✓	✗	6.8 GB	6.8 GB	3.28 it/s	100.0%	100.0%
HiZOO	✓	✗	11.8 GB	11.8 GB	2.22 it/s	59.2%	87.4%
DiZO	✓	✗	7.5 GB	7.5 GB	3.05 it/s	51.8%	55.7%
MeZO LoRA	✓	✓	6.5 GB	6.5 GB	5.56 it/s	74.1%	43.7%
HiZOO LoRA	✓	✓	11.5 GB	11.5 GB	3.70 it/s	46.3%	41.0%
DiZO LoRA	✓	✓	7.2 GB	7.2 GB	4.92 it/s	38.9%	25.9%

Table 11. Memory utilization and speed test on OPT-2.7B on SQuAD dataset (300 tokens per example on average). 🔵: partial gradient-free; ✓: gradient-free; ✗: gradient-based. DiZO†: searching projection with Adam.

Task Type	Gradient Free	LoRA Added	Peak Memory	Averaged Memory	Throughput	#Train Iter.	GPU Hours
FT	✗	✗	73.5 GB	73.5 GB	0.36 it/s	7.5%	27.7%
LoRA	✗	✓	58.5 GB	58.5 GB	0.73 it/s	6.3%	11.5%
DiZO†	🔵	✗	57.8 GB	20.3 GB	1.22 it/s	41.7%	45.5%
DiZO LoRA†	🔵	✓	49.4 GB	19.9 GB	2.44 it/s	31.7%	17.3%
MeZO	✓	✗	8.4 GB	8.4 GB	1.33 it/s	100.0%	100.0%
HiZOO	✓	✗	12.3 GB	13.3 GB	0.97 it/s	66.7%	91.5%
DiZO	✓	✗	9.7 GB	9.7 GB	1.22 it/s	60.0%	65.4%
MeZO LoRA	✓	✓	8.4 GB	8.4 GB	2.80 it/s	73.3%	34.8%
HiZOO LoRA	✓	✓	11.6 GB	12.6 GB	2.10 it/s	56.7%	35.9%
DiZO LoRA	✓	✓	9.6 GB	9.6 GB	2.49 it/s	45.0%	24.0%

To answer the above two questions. We investigate two alternative projection strategies: 1) searching layer-wise ideal learning rate via ZO optimization and then applying the updates, and 2) conducting projection update based on $\boldsymbol{\theta}_{t-1}$ instead of $\boldsymbol{\theta}_0$. Results are illustrated in Table 9, neither approach achieves performance comparable to DiZO; both yield results closer to MeZO in terms of accuracy and required GPU hours.

We attribute this phenomenon to the high noise inherent in each ZO update, which relies on random perturbations and thus produces a highly imprecise update direction. In contrast, DiZO projects the optimization direction between the pre-trained and fine-tuned models, and this direction is supposed to be correct. Otherwise, the entire optimization would fail and the loss would not decrease. Moreover, recent studies suggest that the fine-tuned model is often less robust than their pre-trained version due to catastrophic forgetting (Dong et al., 2021; Oh et al., 2023; Zhai et al., 2023; Wang et al., 2024). Maintaining a connection with the pre-trained model helps robustify the fine-tuning process and mitigate some of the noise introduced by ZO's random perturbations.

D. More Experiment Results

D.1. Memory and Speed Analysis

We present the memory and speed results for OPT-2.7B on the SST-2 and SQuAD datasets in Table 10 and Table 11, respectively. DiZO significantly reduces the number of required iterations while maintaining throughput comparable to MeZO, leading to substantially fewer training GPU hours. In contrast, HiZOO achieves only modest iteration savings and further reduces the throughput of MeZO by approximately 1.5× due to its reliance on second-order information estimation.

As a result, HiZOO offers only a slight improvement over MeZO in terms of training GPU hours. In some cases, such as HiZOO combined with LoRA on SQuAD, it even consumes more training GPU hours than MeZO with LoRA.

D.2. Llama Experiments

To demonstrate the broader applicability of DiZO, we conducted experiments on the Llama-series models. The results for Llama3-3B and Llama3-8B are presented in Table 12 and Table 13, respectively. DiZO consistently outperforms MeZO across both the 3B and 8B Llama models.

However, we observed that ZO LoRA performs poorly with Llama models (including DiZO, MeZO and HiZOO). The loss value remains stagnant, and the resulting accuracy is comparable to or even worse than zero-shot results. We leave it to future work to investigate why ZO LoRA fails with Llama models. We suspect that this limitation may be related to the Group Query Attention (GQA) (Ainslie et al., 2023) mechanism employed in Llama3.

Table 12. Experiments results on Llama3-3B for seven classification datasets and two text generation datasets (with 1000 training samples). Better results between MeZO and DiZO are highlighted in bold.

Task	**SST-2**	**RTE**	**CB**	**BoolQ**	**WSC**	**WIC**	**MultiRC**	**SQuAD**	**DROP**
Task Type	——————————classification——————————							——generation——	
FT	94.2	81.2	91.4	72.2	63.8	65.8	78.2	79.6	40.3
MeZO	88.8	67.4	73.2	**78.0**	56.6	63.4	**64.8**	61.9	27.8
DiZO	**90.0**	**68.2**	**76.7**	76.8	**57.8**	**63.8**	64.2	**63.2**	**29.7**

Table 13. Experiments results on Llama3-8B for seven classification datasets and two text generation datasets (with 1000 training samples). Better results between MeZO and DiZO are highlighted in bold.

Task	**SST-2**	**RTE**	**CB**	**WSC**	**SQuAD**
Task Type	————-classification————-				–generation–
MeZO	90.0	67.8	71.4	60.2	67.0
DiZO	**91.5**	**69.4**	**73.2**	**63.4**	**67.4**

E. Proof

We consider a neural network with L layers (or parameter blocks) and wish to estimate the gradient of some loss function $\mathcal{L}(\boldsymbol{\theta}; \mathcal{B})$ with respect to all parameters $\boldsymbol{\theta}$. We use a two-point finite-difference (zero-order) method with directions drawn from an isotropic distribution. We show below why the *expected* norm-squared of the resulting gradient estimator is *identical* (or follows the same dimension-based law) for each layer/block.

Consider the ℓ-th layer. Its estimator is

$$\widehat{\nabla_{\boldsymbol{\theta}^{(\ell)}}\mathcal{L}} \;=\; \frac{1}{q}\sum_{i=1}^{q}\Big[\underbrace{\frac{\mathcal{L}\big(\boldsymbol{\theta}+\epsilon\boldsymbol{u}_i\big)\;-\;\mathcal{L}\big(\boldsymbol{\theta}-\epsilon\boldsymbol{u}_i\big)}{2\,\epsilon}}_{\Delta_i}\Big]\,\boldsymbol{u}_i^{(\ell)},$$

where Δ_i is the same scalar for *all* blocks. We want

$$\mathbb{E}\Big[\|\widehat{\nabla_{\boldsymbol{\theta}^{(\ell)}}\mathcal{L}}\|^2\Big].$$

Note that:

1. Δ_i does not depend on ℓ; it is a single scalar for each direction i.
2. $\boldsymbol{u}_i^{(\ell)}$ is the sub-vector of $\boldsymbol{u}_i$ associated to the ℓ-th block.
3. $\boldsymbol{u}_i$ is drawn from an *isotropic* distribution in $\mathbb{R}^d$, meaning each coordinate has zero mean, unit variance, and there is no cross-correlation between different coordinates. Thus, different blocks $\boldsymbol{u}_i^{(\ell)}$ and $\boldsymbol{u}_i^{(m)}$ (for $\ell \neq m$) are uncorrelated, and each block $\boldsymbol{u}_i^{(\ell)}$ has an identity covariance in its own subspace $\mathbb{R}^{d_\ell}$.

Hence, when we expand

$$\|\widehat{\nabla_{\boldsymbol{\theta}^{(\ell)}}\mathcal{L}}\|^2 \;=\; \Big\|\frac{1}{q}\sum_{i=1}^{q}\Delta_i\,\boldsymbol{u}_i^{(\ell)}\Big\|^2,$$

the expectation w.r.t. $\{\boldsymbol{u}_i\}$ depends on ℓ *only* through the dimension d_ℓ, not through any other distributional asymmetry. If d_ℓ are the same for all ℓ, then the second moment is *literally* the same across all blocks. If d_ℓ differ, the dependence is only a (known) function of d_ℓ.

In short, **isotropy** ensures that

$$\mathbb{E}\big[\|\widehat{\nabla_{\boldsymbol{\theta}^{(\ell)}}\mathcal{L}}\|^2\big] \quad \text{is the same functional form of } \|\nabla_{\boldsymbol{\theta}^{(\ell)}}\mathcal{L}\|^2 \text{ for each layer } \ell.$$

Therefore, in the simplest scenario where d_ℓ are all the same, each layer gets the *same* second-moment behavior for its gradient estimator.

F. Implementation

The following is an implementation of our "ZO projection learning" in PyTorch.

```
def ZO_Projection_Learning(theta_t, theta_0, Gammas, delta, eta, tau, x):
    """
    Perform Zeroth-order Projection Learning.

    Args:
        theta_t: Current model parameters to be fine-tuned.
        theta_0: Pre-trained model parameters (anchor).
        Gammas: Projection parameters need to be optimized.
        delta: Smoothing parameter.
        eta: Learning rate for projection gradient descent.
        tau: Clipping factor for projection bounds.
        x: Input data for the forward pass.
    """

    # Calculate the L2 norm of the distance gap
    norms = {
        name: torch.norm(param.data - anchor.data)
        for (name, param), anchor in zip(theta_t.named_parameters(), theta_0.parameters())
    }

    # Initialize the projection values
    for name, gamma in Gammas.named_parameters():
        gamma.data = norms[name]

    for i in range(max_iters):
        # Step 1: Perturb and apply projection, then compute loss
        Gammas = PerturbGamma(Gammas, delta)
        ApplyProjection(theta_t, pre_trained, Gammas)
        loss1 = Forward(theta_t, x)
        ReverseProjection(theta_t)  # Reset the parameter before projection

        # Step 2: Reverse and apply projection, then compute loss
        Gammas = PerturbGamma(Gammas, -2 * delta)
        ApplyProjection(theta_t, pre_trained, Gammas)
        loss2 = Forward(theta_t, x)
        ReverseProjection(theta_t)  # Reset the parameter before projection

        # Step 3: Reset projection and compute gradient
        Gammas = PerturbGamma(Gammas, delta)  # Reset projection
        grad = (loss1 - loss2) / (2 * delta)

        # Step 4: Gradient descent with clipping
        for name, gamma in Gammas.named_parameters():
            torch.manual_seed(seed)  # For resampling perturbation
            z = torch.normal(mean=0, std=1, size=gamma.data.size())
            gamma.data = torch.clip(
                gamma.data - eta * grad * z,
                (1 - tau) * norms[name],
                (1 + tau) * norms[name],
            )  # Conduct descent and apply clipping

    return Gammas
```