Enhancing Zeroth-order Algorithm in LLM Fine-tuning with Low-rank Structures

Yiming Chen[1], Yuan Zhang[1], Liyuan Cao[2], Kun Yuan[1], Zaiwen Wen[1]

[1]Peking University [2]Nanjing University

https://github.com/optsuite/LOZO

Outline

1. LLM Fine-tuning
2. Zeroth-order Algorithm in LLMs
3. LOZO Algorithm Framework
4. Theoretical Analysis
5. Numerical Experiments

Introduction to LLM Fine-Tuning

- Large Language Models (LLMs) are pre-trained on massive datasets to understand and generate human language.
- Fine-tuning adapts these pre-trained models to specific tasks or domains, improving their performance on narrower, more targeted use cases.

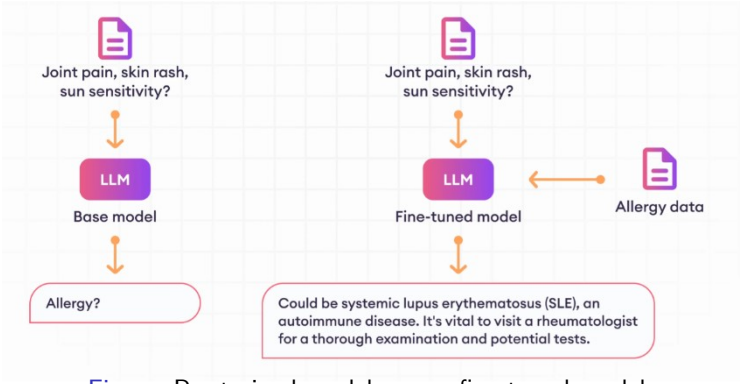


Figure: Pre-trained models vesus fine-tuned models.

Common Methods for LLM Fine-tuning

- **Full Fine-tuning (FT):** Updates all model parameters.
- **Adapter:** Adds lightweight adapter modules to each transformer layer while keeping the original model weights frozen. Reduces training time and memory consumption.
- **LoRA (Low-Rank Adaptation):** Introduces low-rank matrices for efficient adaptation.
- **Prefix-tuning:** Optimizes only a small set of prefix tokens prepended to the input while keeping the model parameters fixed. Efficient for domain-specific adaptations.
- **Prompt-tuning:** Only optimizes a set of soft prompts without changing the model weights. Useful for few-shot learning and highly efficient in terms of memory.

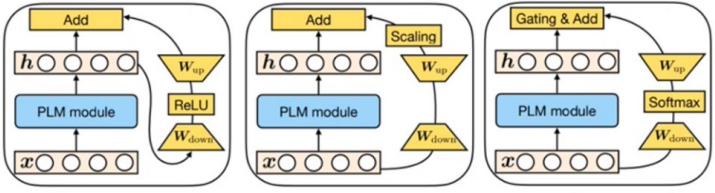


Figure: Illustration of different fine-tuning methods.

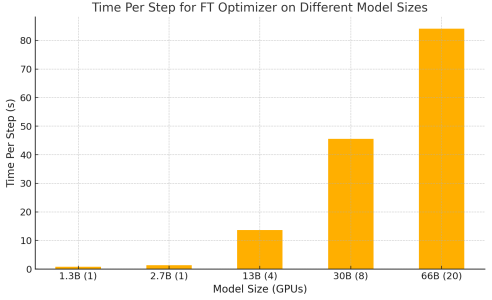
Memory Challenges in LLM Fine-tuning

Optimizer	Memory (GB)	Consumed GPUs
Forward-Grad	138 GB	2 × A100
FO-SGD	161 GB	3 × A100
FO-Adam	257 GB	4 × A100

Hardware	FT	FT-prefix	Inference
1 × A100 (80GB)	2.7B	6.7B	30B
2 × A100 (160GB)	6.7B	13B	66B
4 × A100 (320GB)	13B	30B	66B
8 × A100 (640GB)	30B	66B	175B

Table: The left table shows the memory overhead for LLM fine-tuning with gradient-based algorithms for OPT-13B, while the right table shows GPUs needed for different scale models.

- Fine-tuning LLMs is extensively utilized but requires substantial memory, often necessitating multiple GPUs in practice.
- Due to the communications between GPUs, the time needed for each fine-tuning step increases significantly as the model size grows.



Outline

1 LLM Fine-tuning

2 Zeroth-order Algorithm in LLMs

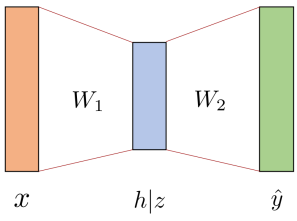
3 LOZO Algorithm Framework

4 Theoretical Analysis

5 Numerical Experiments

Gradient-based Method Needs to Store Activations

The intermediate activations generated during forward propagation must be stored for gradient computation during backward propagation.



Forward	Backward
$h = W_1 x$	$\frac{\partial f}{\partial W_1} = \frac{\partial f}{\partial h} x^T$
$z = \sigma(h)$	$\frac{\partial f}{\partial h} = \frac{\partial f}{\partial z} \odot \nabla\sigma(h)$
$\hat{y} = W_2 z$	$\frac{\partial f}{\partial W_2} = \frac{\partial L}{\partial \hat{y}} z^T, \quad \frac{\partial f}{\partial z} = W_2 \frac{\partial L}{\partial \hat{y}}$
$f = L(\hat{y})$	$\frac{\partial f}{\partial \hat{y}} = \nabla L(\hat{y})$
Store h, z and $\hat{y}$	**Store $\nabla_{W_1} f(W_1)$ and $\nabla_{W_2} f(W_2)$**

If we use gradient-free methods, the memory for storing activations can be avoided!

Finite Difference Based ZO Algorithm

Consider the optimization problem for the LLM fine-tuning task:

$$\min_{\boldsymbol{X}} f(\boldsymbol{X}) := \mathbb{E}_{\xi}[F(\boldsymbol{X}; \xi)],$$

The iterative format of the ZO-SGD algorithm can be represented as follows:

$$\boldsymbol{X}^{t+1} = \boldsymbol{X}^{t} - \alpha \hat{\nabla} F(\boldsymbol{X}^{t}; \xi^{t}),$$

where:

- $\boldsymbol{X}^{t}$ is the current point in the parameter space.
- α is the step size.
- $\hat{\nabla} F(\boldsymbol{X}^{t}; \xi^{t})$ is the approximated gradient of the objective function at $\boldsymbol{X}^{t}$, calculated using finite differences.

Gradient Estimation Scheme

- Randomize vector-wise gradient estimator (RGE):

$$\text{(RGE)} \quad \hat{\nabla} F(\boldsymbol{X}; \xi) := \frac{F(\boldsymbol{X} + \epsilon \boldsymbol{Z}; \xi) - F(\boldsymbol{X} - \epsilon \boldsymbol{Z}; \xi)}{2\epsilon} \boldsymbol{Z},$$

  where $\boldsymbol{Z}$ is sampled from a standard normal distribution.

- Coordinate-wise gradient estimator (CGE):

$$\text{(CGE)} \quad \hat{\nabla} F(\boldsymbol{X}; \xi) := \sum_{i=1}^{d} \frac{F(\boldsymbol{X} + \epsilon \boldsymbol{E}_i; \xi) - F(\boldsymbol{X} - \epsilon \boldsymbol{E}_i; \xi)}{2\epsilon} \boldsymbol{E}_i,$$

  where $\boldsymbol{E}_i$ is a basis vector/matrix with its i-th element set to one and all other elements set to zero.

Memory-efficient ZO-SGD (MeZO) Algorithm

- In [1], the authors first introduce the ZO method for large language model (LLM) fine-tuning, employing a memory-efficient ZO-SGD algorithm with the RGE scheme, referred to as MeZO.
- Compared to gradient-based algorithms, MeZO reduces memory usage as it does not require storing the activations compared with FO algorithms.

Benefits:

- Memory efficiency
- Reduced GPU hours in large models due to less GPU communication.

Drawbacks:

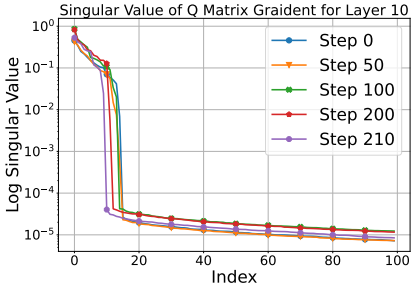
- Slower convergence.
- *Performance gap vs. first-order methods, which we address.*

[1] Malladi S, Gao T, Nichani E, et al. Fine-tuning language models with just forward passes. Advances in Neural Information Processing Systems, 2024, 36.

Outline

1. LLM Fine-tuning
2. Zeroth-order Algorithm in LLMs
3. LOZO Algorithm Framework
4. Theoretical Analysis
5. Numerical Experiments

Low-rank Structure of Gradients



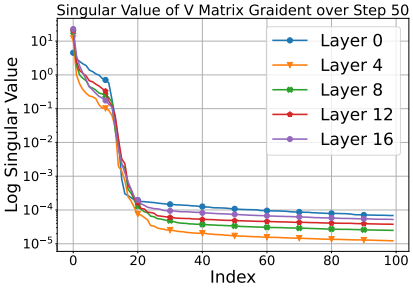


Figure: We conduct experiments to validate the low-rank gradient during the fine-tuning on the OPT-1.3B model with the COPA dataset, where the gradient matrix has dimensions of 2048×2048.

- The true FO gradients generated in LLM fine-tuning exhibit a low-rank structure.
- The RGE scheme produces an estimated gradient that is essentially a projection of the true gradient onto a standard normal random matrix $\boldsymbol{Z}$, which is almost always full rank.

LGE Scheme

- We express $\boldsymbol{X}$ as the set of trainable parameters $\{X_\ell\}_{\ell=1}^{\mathcal{L}}$, where $X_\ell \in \mathbb{R}^{m_\ell \times n_\ell}$, and introduce the following notations:

$$\boldsymbol{U} := \{U_\ell\}_{\ell=1}^{\mathcal{L}}, \boldsymbol{V} := \{V_\ell\}_{\ell=1}^{\mathcal{L}}, \boldsymbol{r} := \{r_\ell\}_{\ell=1}^{\mathcal{L}}.$$

- Low-rank matrix-wise gradient estimator (LGE):

$$\text{(LGE)}\ \hat{\nabla} F(\boldsymbol{X};\xi) := \frac{F(\boldsymbol{X}+\epsilon \boldsymbol{U}\boldsymbol{V}^T;\xi) - F(\boldsymbol{X}-\epsilon \boldsymbol{U}\boldsymbol{V}^T;\xi)}{2\epsilon}(\boldsymbol{U}\boldsymbol{V}^T/\boldsymbol{r}),$$

  where $U_\ell \in \mathbb{R}^{m_\ell \times r_\ell}$ and $V_\ell \in \mathbb{R}^{n_\ell \times r_\ell}$ are sampled from standard normal distributions, with $r_\ell \ll \min(m_\ell, n_\ell)$.

- Using the LGE scheme, the gradient matrix has a rank of at most r_ℓ, effectively capturing the low-rank structure of the FO gradients.

Lazy Update Strategy

- **Vanilla recursion.** For any $t \geq 0$, the vanilla recursion with the LGE scheme is

$$\boldsymbol{X}^{t+1} = \boldsymbol{X}^t - \alpha \hat{\nabla} F(\boldsymbol{X}^t; \xi^t)$$
$$\text{where} \quad \hat{\nabla} F(\boldsymbol{X}^t; \xi^t) = \mathrm{LGE}(\boldsymbol{X}^t, \boldsymbol{U}^t, \boldsymbol{V}^t, \boldsymbol{r}, \epsilon, \xi^t).$$

- **Lazy Update Strategy.** While $\boldsymbol{U}$ is sampled at every iteration t, we only sample $\boldsymbol{V}$ every ν iterations. For $t \in \{k\nu, \ldots, (k+1)\nu - 1\}$,

$$\boldsymbol{X}^{t+1} = \boldsymbol{X}^t - \alpha \hat{\nabla} F(\boldsymbol{X}^t; \xi^t)$$
$$\text{where} \quad \hat{\nabla} F(\boldsymbol{X}^t; \xi^t) = \mathrm{LGE}(\boldsymbol{X}^t, \boldsymbol{U}^t, \boldsymbol{V}^{(k)}, \boldsymbol{r}, \epsilon, \xi^t).$$

Since multiple ZO steps are needed to match a single FO step, our strategy ensures the low-rank structure across accumulated gradient estimators, not just individual steps.

LOZO Algorithm

Algorithm 1: Low-rank ZO-SGD (LOZO)

Input: parameters $\boldsymbol{X}$, step budget T, perturbation scale ϵ, learning rate α, sample interval ν and rank $\{r_\ell\}$.

for $t = 0, \ldots, T-1$ **do**

  foreach $X_\ell \in \boldsymbol{X}$ **do**

    Sample $U_\ell \in \mathbb{R}^{m_\ell \times r_\ell}$ from the standard normal distribution ;

    if $t \bmod \nu = 0$ **then**

      Sample $V_\ell \in \mathbb{R}^{n_\ell \times r_\ell}$ from the standard normal distribution ; `// Resample` V_ℓ

  $\boldsymbol{X} \leftarrow \text{Perturbation}(\boldsymbol{X}, \epsilon, \{U_\ell, V_\ell\})$;

  $F_+ \leftarrow F(\boldsymbol{X}; \xi)$;

  $\boldsymbol{X} \leftarrow \text{Perturbation}(\boldsymbol{X}, -2\epsilon, \{U_\ell, V_\ell\})$;

  $F_- \leftarrow F(\boldsymbol{X}; \xi)$;

  $\boldsymbol{X} \leftarrow \text{Perturbation}(\boldsymbol{X}, \epsilon, \{U_\ell, V_\ell\})$; `// Reset parameters`

  $c \leftarrow (F_+ - F_-)/2\epsilon$; `// Calculate finite difference`

  foreach $X_l \in \boldsymbol{X}$ **do**

    $X_\ell \leftarrow X_\ell - \alpha \cdot c(U_\ell V_\ell^T / r_l)$; `// Update parameters in place`

Function `Perturbation`$(\boldsymbol{X}, \epsilon, \{U_\ell, V_\ell\})$**:**

  foreach $X_\ell \in \boldsymbol{X}$ **do**

    $X_\ell \leftarrow X_\ell + \epsilon U_\ell V_\ell^T$; `// Modify parameters in place`

  return $\boldsymbol{X}$;

LOZO-M Algorithm

- The ZO-SGD with momentum (ZO-SGD-M) can be expressed as follows:

$$\begin{aligned} \boldsymbol{M}^t &= \beta \boldsymbol{M}^{t-1} + (1-\beta)\hat{\nabla} F(\boldsymbol{X}^t; \xi^t), \\ \boldsymbol{X}^{t+1} &= \boldsymbol{X}^t - \alpha \boldsymbol{M}^t. \end{aligned}$$

Compared to vanilla ZO-SGD, ZO-SGD-M introduces additional memory overhead since it requires storing the momentum term, which is proportional to the size of the model.

- By integrating the proposed LOZO algorithm with the momentum technique (LOZO-M), we can effectively mitigate the memory overhead issue.

LOZO-M Algorithm

- The gradient estimator for the LOZO algorithm can then be represented as $\hat{\nabla} F(\boldsymbol{X}^t; \xi^t) = c^t \boldsymbol{U}^t (\boldsymbol{V}^{(k)}/\boldsymbol{r})^T$.
- Since $\boldsymbol{V}^{(k)}$ remains fixed for $t \in \{k\nu, \ldots, (k+1)\nu - 1\}$, we only need to accumulate $c^t \boldsymbol{U}^t$ rather than the full gradient estimator $\hat{\nabla} F(\boldsymbol{X}^t; \xi^t)$ to update the momentum.
- The update rule of LOZO-M is given by:

$$\begin{aligned} \boldsymbol{N}^t &= \beta \boldsymbol{N}^{t-1} + (1-\beta) c^t \boldsymbol{U}^t, \\ \boldsymbol{X}^{t+1} &= \boldsymbol{X}^t - \alpha \boldsymbol{N}^t (\boldsymbol{V}^{(k)}/\boldsymbol{r})^T. \end{aligned}$$

- When updating $\boldsymbol{X}^t$, we compute $\boldsymbol{N}^t (\boldsymbol{V}^{(k)}/\boldsymbol{r})^T$ layer by layer and discard the result immediately after updating the weight of the corresponding layer.

LOZO-M Algorithm

- When $\boldsymbol{V}^{(k)}$ is updated, an additional step is required. Specifically, at $t = (k+1)\nu$, the gradient estimator $\hat{\nabla} F(\boldsymbol{X}^t; \xi^t)$ takes the form $c^t \boldsymbol{U}^t (\boldsymbol{V}^{(k+1)}/\boldsymbol{r})^T$ due to resampling, while the momentum from the previous step is $\boldsymbol{N}^{t-1}(\boldsymbol{V}^{(k)}/\boldsymbol{r})^T$.
- To address this, we project the old momentum onto the new subspace spanned by $\boldsymbol{V}^{(k+1)}$ before updating $\boldsymbol{N}^t$. We compute:

$$\tilde{\boldsymbol{N}}^{t-1} = \arg\min_{\boldsymbol{N}} \|\boldsymbol{N}^{t-1}(\boldsymbol{V}^{(k)})^T - \boldsymbol{N}(\boldsymbol{V}^{(k+1)})^T\|.$$

- Given that $V_\ell^T V_\ell \approx n_\ell I$ for any ℓ , the solution is

$$\tilde{\boldsymbol{N}}^{t-1} \approx \boldsymbol{N}^{t-1}(\boldsymbol{V}^{(k)})^T(\boldsymbol{V}^{(k+1)}/\boldsymbol{n}).$$

Outline

1 LLM Fine-tuning

2 Zeroth-order Algorithm in LLMs

3 LOZO Algorithm Framework

4 Theoretical Analysis

5 Numerical Experiments

Relation with Subspace Minimization

LOZO can be viewed as a subspace minimization method, using ZO to solve subproblems.

- **Random subspace minimization.** Analogous to random coordinate minimization, the random subspace minimization approach is

$$\boldsymbol{B}_k^\star = \arg\min_{\boldsymbol{B}} \{f(\tilde{\boldsymbol{X}}^{(k)} + \boldsymbol{B}(\boldsymbol{V}^{(k)})^T)\}, \ \boldsymbol{V}^{(k)} \text{ follows a normal distribution,}$$
$$\tilde{\boldsymbol{X}}^{(k+1)} = \tilde{\boldsymbol{X}}^{(k)} + \boldsymbol{B}_k^\star(\boldsymbol{V}^{(k)})^T.$$

- **ZO update for the subproblem.** To solve the k-th subproblem, we apply the standard ZO-SGD and iterate for ν steps. The result is then used as an inexact solution:

$$\boldsymbol{B}^{s+1} = \boldsymbol{B}^s - \gamma \hat{\nabla}_{\boldsymbol{B}} F(\tilde{\boldsymbol{X}}^{(k)} + \boldsymbol{B}^s(\boldsymbol{V}^{(k)})^T; \xi^s),$$

where $s = 0, \cdots, \nu - 1$.

Equivalence between ZO Subspace Minimization and LOZO

- By defining $\boldsymbol{Y}^s := \tilde{\boldsymbol{X}}^{(k)} + \boldsymbol{B}^s(\boldsymbol{V}^{(k)})^T$ and applying the update rules for $\boldsymbol{B}$, we can derive the update rule for $\boldsymbol{Y}$, which matches the update rule for $\boldsymbol{X}$ in the LOZO algorithm:

$$\begin{aligned}\boldsymbol{Y}^{s+1} &= \boldsymbol{Y}^s - \gamma \frac{F(\boldsymbol{Y}^s + \epsilon \boldsymbol{U}^s(\boldsymbol{V}^{(k)})^T) - F(\boldsymbol{Y}^s - \epsilon \boldsymbol{U}^s(\boldsymbol{V}^{(k)})^T)}{2\epsilon} \boldsymbol{U}^s(\boldsymbol{V}^{(k)})^T \\ &= \boldsymbol{Y}^s - \alpha \cdot \mathrm{LGE}(\boldsymbol{Y}^s, \boldsymbol{U}^s, \boldsymbol{V}^{(k)}, \boldsymbol{r}, \epsilon, \xi^s), \quad \forall s \in \{0, \cdots, \nu - 1\}.\end{aligned}$$

- Let the iteration sequence generated by the LOZO algorithm be denoted as $\{\boldsymbol{X}^t\}$. Using the relation $\boldsymbol{Y}^{(k,\nu)} = \tilde{\boldsymbol{X}}^{(k+1)}$, we observe that $\boldsymbol{X}^{k\nu} = \tilde{\boldsymbol{X}}^{(k)}$.
- This confirms that the LOZO algorithm operates as a random ZO subspace minimization method.

Convergence Result

Theorem

Let $T = K\nu$. With appropriate choices of the step size α and perturbation scale ϵ, the sequence $\{\boldsymbol{X}^{k\nu}\}$, generated by the LOZO algorithm, converges at the following rate:

$$\frac{1}{K}\sum_{k=0}^{K-1}\mathbb{E}\|\nabla f(\boldsymbol{X}^{k\nu})\|^2 \leq O\left(\sqrt{\frac{\Delta_0 L\tilde{d}\sigma^2}{T}} + \frac{\Delta_0 L d\nu}{T}\right),$$

where $\Delta_0 := f(\boldsymbol{X}^0) - f^*$ represents the initial gap to the optimal value, $\tilde{d} = \sum_{\ell=1}^{\mathcal{L}}(m_\ell n_\ell^2/r_\ell)$, and $d = \sum_{\ell=1}^{\mathcal{L}} m_\ell n_\ell$.

- The results demonstrate that LOZO achieves global convergence.
- The convergence rate depends on d, indicating that more steps are required for fine-tuning large-scale language models.

Outline

1 LLM Fine-tuning

2 Zeroth-order Algorithm in LLMs

3 LOZO Algorithm Framework

4 Theoretical Analysis

5 Numerical Experiments

Performance Comparison on Medium-sized Models

- LOZO and LOZO-M algorithms outperform other ZO methods and achieve performance comparable to FO methods on the RoBERTa-large model.

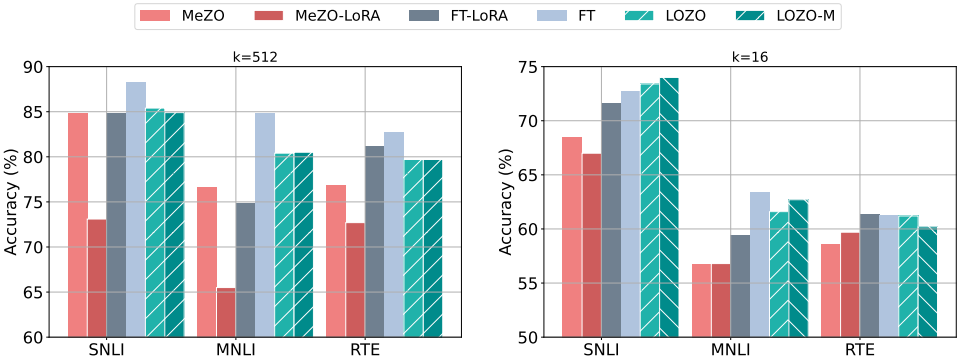


Figure: The figures illustrate the performance of different algorithms on RoBERTa-large across three tasks (SNLI, MNLI, and RTE), with the left panel corresponding to $k = 512$ and the right panel corresponding to $k = 16$.

Performance Comparison on Large-sized Models

- LOZO algorithm also demonstrates performance improvements on large-scale models, including OPT-30B and OPT-66B.

Task	SST-2	RTE	BoolQ	WSC	WiC	SQuAD
30B zero-shot	56.7	52.0	39.1	38.5	50.2	46.5
30B ICL	81.9	**66.8**	66.2	56.7	51.3	78.0
30B MeZO	90.7	64.3	68.2	63.5	56.3	**86.1**
30B LOZO	**92.8**	65.3	**72.3**	**64.4**	**57.2**	85.6
66B zero-shot	57.5	67.2	66.8	43.3	50.6	48.1
66B ICL	89.3	65.3	62.8	52.9	54.9	81.3
66B MeZO	92.0	71.5	73.8	**64.4**	57.8	84.0
66B LOZO	**92.5**	**74.0**	**74.5**	63.5	**59.4**	**85.8**

Table: Experiments on OPT-30B and OPT-66B on the SuperGLUE benchmark. The best results are shown in **bold**.

Memory-efficiency of LOZO and LOZO-M

- LOZO algorithm exhibits superior memory efficiency compared to other algorithms, while the momentum variant, LOZO-M, introduces minimal additional memory overhead.

Task	RTE		MultiRC	
	Memory	**Consumed GPUs**	**Memory**	**Consumed GPUs**
LOZO	27.0 GB	1× A800	26.9 GB	1× A800
LOZO-M	27.4 GB	1× A800	27.3 GB	1× A800
MeZO	27.4 GB	1× A800	27.3 GB	1× A800
MeZO-M	51.7 GB	1× A800	52.1 GB	1× A800
FT-LoRA	79.0 GB	1× A800	102.4 GB	2× A800
FT	250.0 GB	4× A800	315.2 GB	4× A800

Table: Comparison of memory costs for LOZO, MeZO, their momentum variants, and FO methods on OPT-13B.

Many Thanks For Your Attention!