Fine-Tuning Language Models with Just Forward Passes

Sadhika Malladi* **Tianyu Gao*** **Eshaan Nichani** **Alex Damian**

Jason D. Lee **Danqi Chen** **Sanjeev Arora**

Princeton University
{smalladi, tianyug, eshnich, ad27, jasonlee, danqic, arora}@princeton.edu

Abstract

Fine-tuning language models (LMs) has yielded success on diverse downstream tasks, but as LMs grow in size, backpropagation requires a prohibitively large amount of memory. Zeroth-order (ZO) methods can in principle estimate gradients using only two forward passes but are theorized to be catastrophically slow for optimizing large models. In this work, we propose a memory-efficient zeroth-order optimizer (**MeZO**), adapting the classical ZO-SGD method to operate in-place, thereby fine-tuning LMs with *the same memory footprint as inference*. For example, with a single A100 80GB GPU, MeZO can train a 30-billion parameter model, whereas fine-tuning with backpropagation can train only a 2.7B LM with the same budget. We conduct comprehensive experiments across model types (masked and autoregressive LMs), model scales (up to 66B), and downstream tasks (classification, multiple-choice, and generation). Our results demonstrate that (1) MeZO significantly outperforms in-context learning and linear probing; (2) MeZO achieves comparable performance to fine-tuning with backpropagation across multiple tasks, with up to 12× memory reduction and up to 2× GPU-hour reduction in our implementation; (3) MeZO is compatible with both full-parameter and parameter-efficient tuning techniques such as LoRA and prefix tuning; (4) MeZO can effectively optimize non-differentiable objectives (e.g., maximizing accuracy or F1). We support our empirical findings with theoretical insights, highlighting how adequate pre-training and task prompts enable MeZO to fine-tune huge models, despite classical ZO analyses suggesting otherwise.[1]

1 Introduction

Fine-tuning pre-trained language models (LMs) has been the dominant methodology for solving many language tasks [28], adapting to specialized domains [42], or incorporating human instructions and preferences [73]. However, as LMs are scaled up [13, 72], computing gradients for backpropagation requires a prohibitive amount of memory – in our test, up to 12× the memory required for inference – because it needs to cache activations during the forward pass, gradients during the backward pass, and, in the case of Adam [52], also store gradient history (see Section 3.4 for a detailed analysis).

As a result, while it is possible to run inference with a 30-billion (30B) parameter LM on a single Nvidia A100 GPU (with 80GB memory), backpropagation with Adam is feasible only for a 2.7B LM. Parameter-efficient fine-tuning methods (PEFT [46, 57, 54]) update just a fraction of the network

*Equal contribution and corresponding authors.

[1]Our code is available at https://github.com/princeton-nlp/MeZO.

37th Conference on Neural Information Processing Systems (NeurIPS 2023).

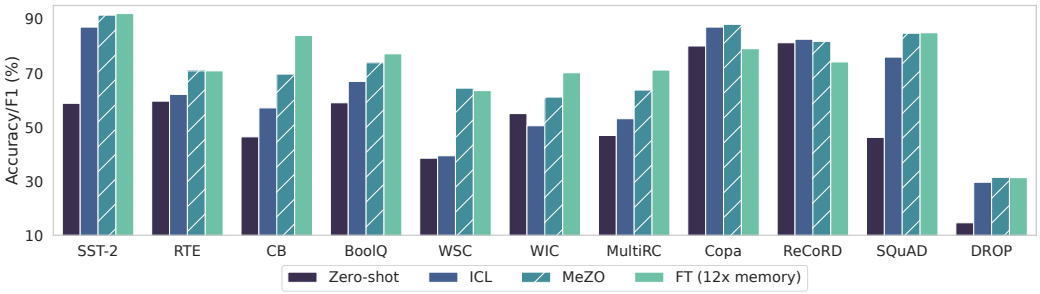

Figure 1: OPT-13B results with zero-shot, in-context learning (ICL), MeZO (we report the best among MeZO/MeZO (LoRA)/MeZO (prefix)), and fine-tuning with Adam (FT). MeZO demonstrates superior results over zero-shot and ICL and performs on par with FT (within 1%) on 7 out of 11 tasks, despite using only 1/12 memory. See Table 1 for detailed numbers and Figure 3 for memory profiling.

parameters, but still need to cache many activations, because the tuned parameters are scattered throughout the model. In our tests, fine-tuning an OPT-13B model with full parameter tuning or PEFT requires $12\times$ and $6\times$ more memory than inference respectively.

In-context learning (ICL [13]) has allowed solving many tasks with a single inference pass, during which the model processes labeled examples (*demonstrations*) in its context and then outputs a prediction on a test example. While this allows for quick adaptation of the model to specific use cases, current models allow a limited context size (and thus, limited demonstrations) and the performance is sensitive to the formatting and choice of demonstrations [60, 66]. ICL can slow with the number of demonstrations, and it often performs worse than fine-tuning of medium-sized models [13].

Backpropagation also cannot optimize non-differentiable criteria, which have gained popularity in fine-tuning LMs according to human preference scores or set safety standards [89, 73]. Typically, these adaptations involve expensive reinforcement learning from human feedback (RLHF [20]).

A classical zeroth-order optimization method, ZO-SGD [88], uses only differences of loss values to estimate the gradients. Thus, in principle, the method can update neural networks with just forward passes, though naive implementation still doubles the memory overhead and classical lower bounds [69, 32] suggest that convergence slows linearly with model size. As such, ZO methods have been applied in deep learning settings to find adversarial examples or tune input embeddings [91, 90] but not to directly optimize large-scale models (see Liu et al. [61] for a survey).

In this work, we propose a memory-efficient zeroth-order optimizer (MeZO), which adapts the classical ZO-SGD algorithm and reduces its memory consumption *to the same as inference*. We apply MeZO to fine-tune large LMs and show that, both empirically and theoretically, MeZO can successfully optimize LMs with billions of parameters. Specifically, our contributions are:

1. In MeZO, we adapt the ZO-SGD algorithm [88] and a number of variants to operate in-place on arbitrarily large models with almost no memory overhead (see Algorithm 1 and Section 2).
2. We conduct comprehensive experiments across model types (masked LM and autoregressive LM), model scales (from 350M to 66B), and downstream tasks (classification, multiple-choice, and generation). MeZO consistently outperforms zero-shot, ICL, and linear probing. Moreover, with RoBERTa-large, MeZO achieves performance close to standard fine-tuning within 5% gap; with OPT-13B, MeZO outperforms or performs comparably to fine-tuning on 7 out of 11 tasks, despite requiring roughly $12\times$ less memory (Figure 1 and Section 3). In our implementation, MeZO requires only half as many GPU-hours as Adam fine-tuning for a 30B model (see Appendix F.6).
3. We demonstrate MeZO's compatibility with full-parameter tuning and PEFT (e.g., LoRA [46] and prefix-tuning [57]) in Section 3.
4. Further exploration showcases that MeZO can optimize non-differentiable objectives such as accuracy or F1 score, while still requiring only the same memory as inference (Section 3.3).
5. Our theory suggests that adequate pre-training ensures the per-step optimization rate (Theorem 1) and global convergence rate (Lemma 3) of MeZO depend on a certain condition number of the landscape (i.e., the local effective rank, see Assumption 1) instead of numbers of parameters. This

Algorithm 1: MeZO

Require: parameters $\boldsymbol{\theta} \in \mathbb{R}^d$, loss $\mathcal{L} : \mathbb{R}^d \to \mathbb{R}$, step budget T, perturbation scale ϵ, batch size B, learning rate schedule $\{\eta_t\}$

for $t = 1, ..., T$ **do**
  Sample batch $\mathcal{B} \subset \mathcal{D}$ and random seed s
  $\boldsymbol{\theta} \leftarrow$ `PerturbParameters`$(\boldsymbol{\theta}, \epsilon, s)$
  $\ell_+ \leftarrow \mathcal{L}(\boldsymbol{\theta}; \mathcal{B})$
  $\boldsymbol{\theta} \leftarrow$ `PerturbParameters`$(\boldsymbol{\theta}, -2\epsilon, s)$
  $\ell_- \leftarrow \mathcal{L}(\boldsymbol{\theta}; \mathcal{B})$
  $\boldsymbol{\theta} \leftarrow$ `PerturbParameters`$(\boldsymbol{\theta}, \epsilon, s)$ ▷ `Reset parameters before descent`
  `projected_grad` $\leftarrow (\ell_+ - \ell_-)/(2\epsilon)$
  Reset random number generator with seed s ▷ `For sampling` z
  for $\theta_i \in \boldsymbol{\theta}$ **do**
    $z \sim \mathcal{N}(0, 1)$
    $\theta_i \leftarrow \theta_i - \eta_t *$ `projected_grad` $* z$
  end
end

Subroutine `PerturbParameters`$(\boldsymbol{\theta}, \epsilon, s)$
  Reset random number generator with seed s ▷ `For sampling` z
  for $\theta_i \in \boldsymbol{\theta}$ **do**
    $z \sim \mathcal{N}(0, 1)$
    $\theta_i \leftarrow \theta_i + \epsilon z$ ▷ `Modify parameters in place`
  end
  return $\boldsymbol{\theta}$

result is in sharp contrast to existing ZO lower bounds [69, 32] suggesting that the convergence rate can slow proportionally to the number of parameters (Section 4).

2 Zeroth-order optimization

Zeroth-order (ZO) optimizers have long been studied in the context of convex and strongly convex objectives. In the following, we first introduce a classical ZO gradient estimator, SPSA (Definition 1 [88]) and the corresponding SGD algorithm, ZO-SGD (Definition 2). Then we describe MeZO, our in-place implementation that requires the same memory as inference in Section 2.1 and Algorithm 1. We highlight that SPSA can also be used in more complex optimizers, such as Adam, and we provide memory-efficient implementations for those algorithms too (Section 2.2).

Consider a labelled dataset $\mathcal{D} = \{(\boldsymbol{x}_i, \boldsymbol{y}_i)\}_{i \in [|\mathcal{D}|]}$ and a minibatch $\mathcal{B} \subset \mathcal{D}$ of size B, we let $\mathcal{L}(\boldsymbol{\theta}; \mathcal{B})$ denote the loss on the minibatch. We introduce a classical ZO gradient estimate in this setting.[2]

Definition 1 (Simultaneous Perturbation Stochastic Approximation or SPSA [88])**.** *Given a model with parameters $\boldsymbol{\theta} \in \mathbb{R}^d$ and a loss function $\mathcal{L}$, SPSA estimates the gradient on a minibatch $\mathcal{B}$ as*

$$\widehat{\nabla}\mathcal{L}(\boldsymbol{\theta}; \mathcal{B}) = \frac{\mathcal{L}(\boldsymbol{\theta} + \epsilon \boldsymbol{z}; \mathcal{B}) - \mathcal{L}(\boldsymbol{\theta} - \epsilon \boldsymbol{z}; \mathcal{B})}{2\epsilon} \boldsymbol{z} \approx \boldsymbol{z}\boldsymbol{z}^\top \nabla \mathcal{L}(\boldsymbol{\theta}; \mathcal{B}) \tag{1}$$

where $\boldsymbol{z} \in \mathbb{R}^d$ with $\boldsymbol{z} \sim \mathcal{N}(0, \boldsymbol{I}_d)$ and ϵ is the perturbation scale*. The n-SPSA gradient estimate averages $\widehat{\nabla}\mathcal{L}(\boldsymbol{\theta}; \mathcal{B})$ over n randomly sampled $\boldsymbol{z}$.*

SPSA requires only *two forward passes* through the model to compute the gradient estimate (for n-SPSA, each estimate requires $2n$ forward passes). As $\epsilon \to 0$, the SPSA estimate can be understood as a rank-1 reconstruction of the gradient. During training, n can be treated as a hyperparameter and follow a schedule [11, 15], though in cursory experiments (Appendix A), $n = 1$ is the most efficient.

[2]The original SPSA algorithm [88] perturbs the model by $1/\boldsymbol{z}$ and thus requires that $\boldsymbol{z}$ has finite inverse moments, precluding the choice of $\boldsymbol{z}$ as Gaussian. $1/\boldsymbol{z}$ is very large with high probability for a zero-mean Gaussian $\boldsymbol{z}$, so we adopt the standard in many theoretical [70, 32] and empirical [64] works and perturb the parameters by $\boldsymbol{z}$ with $\boldsymbol{z}$ as a Gaussian random variable.

We use $n = 1$ as the default. It is widely known that the SPSA estimate can be used to replace the backpropagation gradient in any optimizer such as SGD.

Definition 2 (ZO-SGD)**.** *ZO-SGD is an optimizer with learning rate η that updates parameters as $\boldsymbol{\theta}_{t+1} = \boldsymbol{\theta}_t - \eta\widehat{\nabla}\mathcal{L}(\boldsymbol{\theta}; \mathcal{B}_t)$ where $\mathcal{B}_t$ is the minibatch at time t and $\widehat{\nabla}\mathcal{L}$ is the SPSA gradient estimate.*

2.1 Memory-efficient ZO-SGD (MeZO)

The vanilla ZO-SGD algorithm costs twice the memory of inference, as it needs to store $\boldsymbol{z} \in \mathbb{R}^d$. We propose a memory-efficient implementation of ZO-SGD called **MeZO**, as illustrated in Algorithm 1. At each step, we first sample a random seed s, and then for each of $\boldsymbol{z}$'s four uses in Algorithm 1, we reset the random number generator by s and *resample* the relevant entry of $\boldsymbol{z}$. Using this in-place implementation, MeZO has a memory footprint equivalent to the inference memory cost.

We note that Algorithm 1 describes perturbing each parameter separately, which may be time-consuming for large models. In practice, we can save time by perturbing an entire weight matrix instead of each scalar independently. This incurs an additional memory cost as large as the largest weight matrix; usually, this is the word embedding matrix (e.g., 0.86GB for OPT-66B).

Storage Efficiency of MeZO. Parameter-efficient fine-tuning (PEFT) techniques fine-tune just a fraction of the network parameters and have thus been proposed as a way to reduce the storage costs of fine-tuned model checkpoints. Fine-tuning with MeZO reduces the storage cost of the resulting checkpoint far more than popular PEFT techniques (e.g., LoRA [46] and prefix tuning [57]). We reconstruct the MeZO trajectory using a single seed, which spawns step-wise seeds to sample $\boldsymbol{z}$, and the `projected_grad` at each step.[3] As such, for fine-tuning a 66B model, MeZO requires saving the seed plus 20,000 (steps) $\times$ 2 bytes, which is less than 0.1MB. LoRA fine-tunes 19M parameters and requires 38MB storage, and prefix tuning fine-tunes 6M parameters and requires 12MB storage.

2.2 MeZO extensions

We note that SPSA is a popular ZO gradient estimator but not the only one. Many one-point gradient estimators have been proposed in past works [34, 87, 95], and using such estimators in place of SPSA would halve the training time. However, cursory experiments with one such promising estimator [113] reveal that these are not as efficient as SPSA when fixing the number of forward passes (Appendix B.5). As such, we implement MeZO with the SPSA estimator.

MeZO can also be combined with other gradient-based optimizers, including SGD with momentum or Adam. Though naive implementation would require additional memory to store the gradient moment estimates, MeZO-momentum and MeZO-Adam alleviate such overhead by recomputing the moving average of the gradients using saved past losses and $\boldsymbol{z}$ (see Appendix B for a full discussion).

We also note that all of the coordinates of the SPSA gradient estimate have the same scale, but deep Transformers can have gradients of different scales for each layer [59, 61]. As such, we draw inspiration from layerwise adaptive optimizers [109, 110] to design several MeZO variants. Cursory experiments showed that these algorithms are not more efficient (in terms of forward passes), but we nevertheless present them as potential optimizers for more complex objectives. See Appendix B.

Forward Auto-Differentiation Note that $\boldsymbol{z}^\top \nabla\mathcal{L}(\boldsymbol{\theta}; \mathcal{B})$ is a Jacobian-vector product (JVP), which can be computed in parallel with an inference pass with excess memory consumption equivalent to that of the largest activation in the network [40]. In this case, $\boldsymbol{z}$ must be stored on the GPU in order to construct the gradient estimate, so this procedure requires slightly more than two times the memory needed for inference. We analyze this algorithm in detail in Appendix D. Note that using a non-zero ϵ in SPSA, which is not possible through the JVP method, may boost generalization by promoting a sharpness-minimizing term. Past works (e.g., Baydin et al. [9]) have also studied JVP-based training but achieved limited empirical success.

[3] Note that this reconstruction requires no additional forward passes through the model and no access to the data used during fine-tuning, since `projected_grad` implicitly encodes this information.

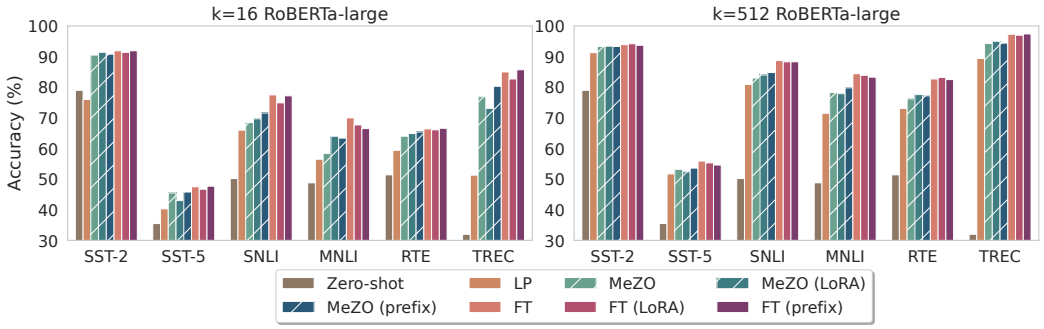

Figure 2: Experiments on RoBERTa-large. We report zero-shot, linear probing (LP), and MeZO and fine-tuning (FT) with full parameter, LoRA, and prefix-tuning. MeZO outperforms zero-shot and LP and approaches FT (within 5% for $k = 512$) with much less memory. Detailed numbers in Table 18.

3 Experiments

Preliminary experiments (Appendix A) show that MeZO only works when using prompts [13, 84, 35]. Past works [83, 67] have demonstrated how the inclusion of a suitable prompt ensures the fine-tuning objective is closely related to the pre-training one. In Section 4, we extend these ideas to show how using a simple prompt simplifies the fine-tuning optimization procedure, thereby enabling zeroth order methods to work efficiently. All experiments below use prompts detailed in Appendix E.2. All fine-tuning with backpropagation (FT) experiments follow convention and use Adam, though we also report results when performing FT with SGD in Appendix F.

We conduct comprehensive experiments on both medium-sized masked LMs (RoBERTa-large, 350M [65]) and large autoregressive LMs (OPT-13B, 30B, 66B [112]) in few-shot and many-shot settings with prompts. We also explore both full-parameter tuning and PEFT including LoRA [46] and prefix-tuning [57] (see Appendix E.5 for details). We compare MeZO with zero-shot, in-context learning (ICL), linear-probing (LP), and fine-tuning with Adam (FT). MeZO uses substantially less memory than FT but requires significantly more training steps.

We first show that MeZO improves substantially over zero-shot, ICL, and LP across model types, sizes, and task types. Moreover, MeZO performs comparably to FT over a number of tasks, while drastically reducing the memory cost by, for example, 12× on OPT-13B. Further experiments demonstrate that MeZO can optimize non-differentiable objectives, such as accuracy and F1 score (Section 3.3). We compare the memory consumption of ICL, FT, LP, and MeZO in Figures 3 and 4.

3.1 Medium-sized masked language models

We conduct experiments with RoBERTa-large on sentiment classification, natural language inference, and topic classification tasks. We follow past works [35, 67] in studying the few-shot and many-shot settings, sampling k examples per class for $k = 16$ and $k = 512$ (details in Appendix E). We run MeZO for 100K steps and fine-tuning for 1000 steps, noting that one MeZO step is substantially faster than one fine-tuning step (see Appendix F.6 for a comparison). We summarize the results from Figure 2 and Table 18 below.

MeZO works significantly better than zero-shot, linear probing, and other memory-equivalent methods. On all six diverse tasks, MeZO can optimize the pre-trained model and consistently perform better than zero-shot and linear probing. We also show for several tasks that MeZO can outperform another ZO algorithm, BBTv2 [90], by up to 11% absolute (Appendix F.4).[4]

With enough data, MeZO achieves comparable performance (up to 5% gap) to FT. MeZO achieves close-to-fine-tuning performance on $k = 16$, with some tasks only having 2% gaps. When using $k = 512$ data, the gap between MeZO and FT further reduced to within 5% across all tasks.

MeZO works well on both full-parameter tuning and PEFT. Full-parameter tuning (MeZO) and PEFT (MeZO with LoRA and prefix-tuning) achieve comparable performance, while MeZO (prefix)

[4] BBTv2 can only train low-dimensional projected prefixes instead of the full model.

Task Task type	SST-2	RTE	CB	BoolQ	WSC	WIC	MultiRC	COPA	ReCoRD	SQuAD	DROP
	classification							– multiple choice –		— generation —	
Zero-shot	58.8	59.6	46.4	59.0	38.5	55.0	46.9	80.0	81.2	46.2	14.6
ICL	87.0	62.1	57.1	66.9	39.4	50.5	53.1	87.0	**82.5**	75.9	29.6
LP	**93.4**	68.6	67.9	59.3	63.5	60.2	63.5	55.0	27.1	3.7	11.1
MeZO	91.4	66.1	67.9	67.6	63.5	**61.1**	60.1	**88.0**	81.7	**84.7**	30.9
MeZO (LoRA)	89.6	67.9	66.1	**73.8**	**64.4**	59.7	61.5	84.0	81.2	83.8	**31.4**
MeZO (prefix)	90.7	**70.8**	**69.6**	73.1	60.6	59.9	**63.7**	87.0	81.4	84.2	28.9
FT (12x memory)	92.0	70.8	83.9	77.1	63.5	70.1	71.1	79.0	74.1	84.9	31.3

Table 1: Experiments on OPT-13B (with 1000 examples). ICL: in-context learning; LP: linear probing; FT: full fine-tuning with Adam. MeZO outperforms zero-shot, ICL, and LP across the board, and achieves comparable (within 1%) or better performance than FT on 7 out of 11 tasks.

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/MeZO (prefix)	**90.6**	**72.6**	**73.5**	**63.5**	**59.1**	**85.2**
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/MeZO (prefix)	**93.6**	66.4	**73.7**	**63.5**	**58.9**	**85.0**

Table 2: Experiments on OPT-30B and OPT-66B (with 1000 examples). We report the best of MeZO and MeZO (prefix). See Appendix F.2 for more results. We see that on most tasks MeZO effectively optimizes up to 66B models and outperforms zero-shot and ICL.

sometimes outperforms MeZO. We also show in Appendix F.3 that the three variants converge at similar rates, agreeing with our theory in Section 4, which shows that MeZO converges at a rate independent of the number of parameters being optimized.

We show additional results with more FT and MeZO variants in Appendix F.1. We see that (1) ZO-Adam sometimes outperforms ZO-SGD but is not consistent across tasks; (2) LP and then MeZO, as suggested for fine-tuning [53], can sometimes improve the performance.

3.2 Large autoregressive language models

With the promising results from RoBERTa-large, we extend MeZO to the OPT family [112], on a scale of 13B (Table 1), 30B, and 66B (Table 2). We select both SuperGLUE [98] tasks[5] (including classification and multiple-choice) and generation tasks. We randomly sample 1000, 500, and 1000 examples for training, validation, and test, respectively, for each datset. We run MeZO for 20K steps and fine-tuning for 5 epochs, or 625 steps, noting that each step of MeZO is substantially faster than fine-tuning (see Appendix F.6 for a comparison). Please refer to Appendix E for details. Table 1 yields the following observations.

MeZO outperforms memory-equivalent methods and closely approaches fine-tuning results. We see that on a 13B-parameter scale, MeZO and its PEFT variants outperform zero-shot, ICL, and LP across almost all tasks. When comparing to FT, which costs 12× more memory (Section 3.4), MeZO achieves comparable (within 1%) or better performance on 7 out of the 11 tasks.

MeZO exhibits strong performance across classification, multiple-choice, and generation tasks. We investigate MeZO on generation tasks, which are regarded as more intricate than classification or multiple-choice tasks. We evaluate on two question answering datasets, SQuAD [80] and DROP [31]. We use teacher forcing for training and greedy decoding for inference (details in Appendix E).

Table 1 shows that, on all generation tasks, MeZO outperforms zero-shot, ICL, and LP, and achieves comparable performance to FT. Considering that many applications of fine-tuning LMs – including instruction tuning or domain adaptation – target generation tasks, our results underscore the potential of MeZO as a memory-efficient technique to optimize large LMs for realistic and exciting applications.

[5]We also include SST-2, which is a simple sentiment classification task that we use for development.

Model	RoBERTa-large (350M)				OPT-13B
Task	**SST-2**	**SST-5**	**SNLI**	**TREC**	**SQuAD**
Zero-shot	79.0	35.5	50.2	32.0	46.2
Cross entropy (FT)	93.9	55.9	88.7	97.3	84.2
Cross entropy (MeZO)	93.3	53.2	83.0	94.3	84.7
Accuracy/F1 (MeZO)	92.7	48.9	82.7	68.6	78.5

Table 3: MeZO with non-differentiable objectives. For classification ($k = 512$), we use MeZO with full-parameter and optimize accuracy; for SQuAD (1,000 examples), we use MeZO (prefix) and F1.

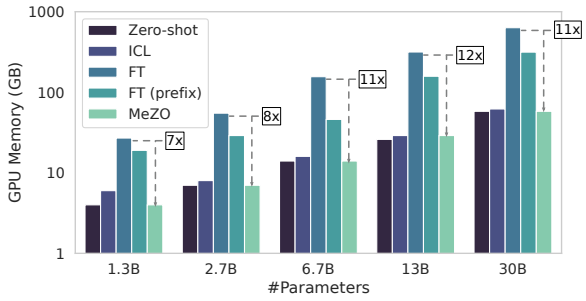

Figure 3: GPU memory consumption with different OPT models and tuning methods on MultiRC (400 tokens per example on average).

Hardware	Largest OPT that can fit		
	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†

Figure 4: Largest OPT models that one can tune with specific hardwares and algorithms. † : projected results without actual testing.

MeZO scales up to 66 billion parameter models. We demonstrate the efficacy of MeZO on even larger models, up to 66B, in Table 2. While directly fine-tuning models at such scales is extremely costly (Section 3.4), MeZO can effectively optimize these models and outperform zero-shot and ICL.

3.3 Training with non-differentiable objectives

We demonstrate the efficacy of MeZO for optimizing non-differentiable objectives through initial experiments. Accuracy and F1 are used as the respective objectives (details in Appendix E.6). Table 3 reveals that MeZO with accuracy/F1 successfully optimizes LMs with superior performance to zero-shot. Although minimizing cross entropy results in stronger performance, these preliminary findings highlight the promising potential of applying MeZO to optimize non-differentiable objectives without clear differentiable surrogates, such as human preferences [73].

3.4 Memory usage and wall-clock time analysis

In this section we profile the memory usage of zero-shot, ICL, FT, FT (prefix), and MeZO. We test OPT models of various sizes with Nvidia A100 GPUs (80GB memory) on MultiRC (average #tokens=400), and report the peak GPU memory consumption (details in Appendix E.7).

As shown in Figure 3 (refer to Appendix F.5 for detailed numbers), MeZO exhibits the same memory consumption as zero-shot while offering memory savings of up to 12 times compared to standard FT and 6 times compared to FT (prefix). This advantage enables training larger models within a fixed hardware budget, as illustrated in Figure 4. Specifically, using a single A100 GPU, MeZO allows for tuning a model that is 11 times larger than what is feasible with FT.

In Appendix F.6, we compare the wall-clock time efficiencies of our implementations of MeZO and Adam fine-tuning. MeZO achieves 7.74× per-step speedup and requires 8× fewer GPUs with a 30B model, but takes more steps to converge. Overall, MeZO only requires half as many GPU-hours to fine-tune a 30B model compared to full-parameter fine-tuning. The wall-clock benefit of MeZO is not inherent to the algorithm and is highly dependent on the implementation. We primarily provide this information as a demonstration that MeZO does not take a prohibitively long time to run.

The above measurements are dependent on the computing infrastructure. In Appendix C, we compare the theoretical time-memory tradeoff of MeZO and backpropagation and find that MeZO is always more memory-efficient than backpropagation and is often more time-efficient. The above analyses also do not consider recent advances (e.g., gradient checkpointing [18], FlashAttention [23], and quantization [27]). We leave investigating the how MeZO works with these methods to future work.

4 Theory

Our theoretical analysis highlights why MeZO can optimize large LMs, although a number of classical results [69, 47, 79, 3, 70] suggest that optimization should be catastrophically slow when training so many parameters. The inclusion of a simple prompt is crucial for MeZO to succeed (Appendix A). Past works [83, 67] have suggested that including such a prompt ensures that the fine-tuning objective is closely related to the pre-training one. As such, here, we make the assumption that the model has already been trained for many steps on the fine-tuning objective, which implies that the loss landscape exhibits favorable conditions (Assumption 1). Then, we derive a convergence rate independent of the number of parameters. We show that the loss decreases per step at a rate independent of the parameter dimension d (Theorem 1), and that, under stronger conditions, the algorithm converges in time independent of d (Lemma 3). Together, these results imply that MeZO is not catastrophically slower than SGD when fine-tuning.[6] For ease of illustration, we assume that $\boldsymbol{z}$ is sampled from a sphere with radius $\sqrt{d}$, and in Appendix G.2, we derive the rate for a general Gaussian $\boldsymbol{z}$, which was used in the experiments.

We follow classical analyses of SGD and replace the minibatch gradient estimate with SPSA (Definition 1). Consider the minibatch SGD update $\boldsymbol{\theta}_{t+1} \leftarrow \boldsymbol{\theta}_t - \eta \nabla \mathcal{L}(\boldsymbol{\theta}; \mathcal{B}_t)$ where $\mathcal{B}_t$ is a minibatch drawn uniformly from $\mathcal{D}^B$. Crucially, the SGD minibatch gradient estimate is unbiased.

Definition 3 (Unbiased Gradient Estimate)**.** *Any minibatch gradient estimate $\boldsymbol{g}(\boldsymbol{\theta}, \mathcal{B})$ is said to be unbiased if $\mathbb{E}[\boldsymbol{g}(\boldsymbol{\theta}, \mathcal{B})] = \nabla \mathcal{L}(\boldsymbol{\theta})$.*

4.1 Per-step analysis

The classical descent lemma uses a Taylor expansion to study how SGD reduces the loss at each optimization step. It highlights that when the gradient covariance is large, the maximum possible decrease in loss at each optimization step is small, thereby resulting in slower optimization.

Lemma 1 (Descent Lemma)**.** *Let $\mathcal{L}(\boldsymbol{\theta})$ be ℓ-smooth.[7] For any unbiased gradient estimate $\boldsymbol{g}(\boldsymbol{\theta}, \mathcal{B})$,*

$$\mathbb{E}[\mathcal{L}(\boldsymbol{\theta}_{t+1}) \mid \boldsymbol{\theta}_t] - \mathcal{L}(\boldsymbol{\theta}_t) \leq -\eta \left\|\nabla \mathcal{L}(\boldsymbol{\theta}_t)\right\|^2 + \frac{1}{2}\eta^2 \ell \cdot \mathbb{E}[\|\boldsymbol{g}(\boldsymbol{\theta}, \mathcal{B}_t)\|^2]. \tag{2}$$

The descent lemma highlights the importance of the gradient norm, which we derive for MeZO below.

Lemma 2. *Let $\mathcal{B}$ be a random minibatch of size B. Then, the gradient norm of MeZO is*

$$\mathbb{E}_x\left[\left\|\widehat{\nabla} \mathcal{L}(\boldsymbol{\theta}; \mathcal{B})\right\|^2\right] = \frac{d+n-1}{n} \mathbb{E}\left[\left\|\nabla \mathcal{L}(\boldsymbol{\theta}; \mathcal{B})\right\|^2\right].$$

where n is the number of $\boldsymbol{z}$ sampled in n-SPSA (Definition 1) and d is the number of parameters.

Thus, in the usual case where $n \ll d$, MeZO has a much larger gradient norm than SGD.[8] The descent lemma also shows that to guarantee loss decrease, one needs to choose the learning rate as

$$\eta \leq \frac{2\left\|\nabla \mathcal{L}(\boldsymbol{\theta}_t)\right\|^2}{\ell \cdot \mathbb{E}[\|\boldsymbol{g}(\boldsymbol{\theta}, \mathcal{B})\|^2]} \quad \xRightarrow{\text{Lemma 2}} \quad \eta_{\text{ZO}} = \frac{n}{d+n-1}\eta_{\text{SGD}} \tag{3}$$

where η_{ZO} and η_{SGD} are the maximum permissible learning rates for MeZO and SGD respectively. Thus we see that without any further assumptions, MeZO can slow optimization by decreasing the largest permissible learning rate by a factor of d. Moreover, MeZO reduces the loss decrease that can be obtained at each step and, as a consequence, slows convergence by a factor of d as well.

Surprisingly, our experiments show that MeZO can quickly optimize pre-trained models with billions of parameters, and reducing the number of tuned parameters via PEFT techniques does not substantially accelerate optimization (Appendix F.3). We attribute these phenomena to the Hessian of the loss exhibiting small local effective rank. It is prohibitively expensive to directly measure the effective rank of the Hessian of a large LM on a reasonably sized dataset. However, many previous works have shown that the Hessian of the loss for deep neural networks trained by SGD has remarkably low

[6] Section 3 uses the standard choice of Adam for FT; we provide SGD experiments in Appendix F.1.

[7] This is satisfied for the standard cross-entropy objective.

[8] All of our experiments use $n = 1$.

effective rank [74, 75, 36, 107, 105, 82]. In particular, the bulk of the spectrum concentrates around 0 with only a small number of outliers, and the number of these outliers is an upper bound on the effective rank. In addition, prior works [4, 56] have demonstrated that LM fine-tuning can occur in a very low dimensional subspace (< 200 parameters), which further supports the below assumption. We formalize the assumption on the effective rank below. In particular, we require an upper bound on the Hessian in a neighborhood around the current iterate to have effective rank at most r.

Assumption 1 (Local r-effective rank)**.** *Let $G(\boldsymbol{\theta}_t) = \max_{(\boldsymbol{x},\boldsymbol{y})\in\mathcal{D}} \|\nabla\mathcal{L}(\boldsymbol{\theta}_t; \{(\boldsymbol{x},\boldsymbol{y})\})\|$. There exists a matrix $\boldsymbol{H}(\boldsymbol{\theta}_t) \preceq \ell \cdot \boldsymbol{I}_d$ such that:*

1. *For all $\boldsymbol{\theta}$ such that $\|\boldsymbol{\theta} - \boldsymbol{\theta}_t\| \leq \eta d G(\boldsymbol{\theta}_t)$, we have $\nabla^2\mathcal{L}(\boldsymbol{\theta}) \preceq \boldsymbol{H}(\boldsymbol{\theta}_t)$.*
2. *The effective rank of $\boldsymbol{H}(\boldsymbol{\theta}_t)$, i.e $\operatorname{tr}(\boldsymbol{H}(\boldsymbol{\theta}_t))/\|\boldsymbol{H}(\boldsymbol{\theta}_t)\|_{op}$, is at most r.*

Under this assumption, we show that the convergence rate of ZO-SGD does not depend on the number of parameters. Instead, the slowdown factor only depends on the effective rank of the Hessian.

Theorem 1 (Dimension-Free Rate)**.** *Assume the loss exhibits local r-effective rank (Assumption 1). If $\boldsymbol{\theta}_{t+1} = \boldsymbol{\theta}_t - \eta_{ZO}\widehat{\nabla}\mathcal{L}(\boldsymbol{\theta}_t; \mathcal{B})$ is a single step of ZO-SGD using the n-SPSA estimate with a minibatch of size B, then there exists a $\gamma = \Theta(r/n)$ such that the expected loss decrease can be bounded as*

$$\mathbb{E}[\mathcal{L}(\boldsymbol{\theta}_{t+1}) \mid \boldsymbol{\theta}_t] - \mathcal{L}(\boldsymbol{\theta}_t) \leq -\eta_{ZO}\|\nabla\mathcal{L}(\boldsymbol{\theta}_t)\|^2 + \frac{1}{2}\eta_{ZO}^2\ell\cdot\gamma\cdot\mathbb{E}[\|\nabla\mathcal{L}(\boldsymbol{\theta};\mathcal{B})\|^2] \tag{4}$$

By applying Equation (3), we can directly compare to the SGD descent lemma.

Corollary 1. *Choosing the learning rate $\eta_{ZO} = \gamma^{-1}\cdot\eta_{SGD}$, ZO-SGD obtains a loss decrease of*

$$\mathbb{E}[\mathcal{L}(\boldsymbol{\theta}_{t+1}) \mid \boldsymbol{\theta}_t] - \mathcal{L}(\boldsymbol{\theta}_t) \leq \frac{1}{\gamma}\cdot\left[-\eta_{SGD}\|\nabla\mathcal{L}(\boldsymbol{\theta}_t)\|^2 + \frac{1}{2}\eta_{SGD}^2\ell\cdot\mathbb{E}[\|\nabla\mathcal{L}(\boldsymbol{\theta};\mathcal{B})\|^2]\right]. \tag{5}$$

Here we see that comparing to SGD, the slowdown factor of ZO-SGD scales with the local effective rank r, which we argue is much smaller than the number of parameters d. The above analysis focuses on how much ZO-SGD and SGD decrease the loss at each step. Below, we show that under stronger assumptions about the loss landscape, we can obtain rates for how quickly the ZO-SGD algorithm converges to an optimal value.

4.2 Global convergence analysis

We show that the global convergence rate also slows by a factor proportional to the local effective rank under stronger assumptions about the loss landscape. We assume that the landscape obeys the classical PL inequality: the gradient norm grows quadratically with the suboptimality of the iterate.

Definition 4 (PL Inequality)**.** *Let $\mathcal{L}^* = \min_{\boldsymbol{\theta}}\mathcal{L}(\boldsymbol{\theta})$. The loss $\mathcal{L}$ is μ-PL if, for all $\boldsymbol{\theta}$, $\frac{1}{2}\|\nabla\mathcal{L}(\boldsymbol{\theta})\|^2 \geq \mu(\mathcal{L}(\boldsymbol{\theta}) - \mathcal{L}^*)$.*

The PL inequality is not as strong as assuming that optimization exhibits kernel-like dynamics, but it ensures that the landscape is amenable to analysis [50]. In addition to the PL inequality, we assume the trace of the gradient covariance is bounded, so noise does not disrupt the trajectory too drastically.

Definition 5 (Gradient Covariance)**.** *The SGD gradient estimate on a minibatch of size B has covariance $\boldsymbol{\Sigma}(\boldsymbol{\theta}) = B(\mathbb{E}\left[\nabla\mathcal{L}(\boldsymbol{\theta};\mathcal{B})\nabla\mathcal{L}(\boldsymbol{\theta};\mathcal{B})^\top\right] - \nabla\mathcal{L}(\boldsymbol{\theta})\nabla\mathcal{L}(\boldsymbol{\theta})^\top)$.*

As we show in Appendix G.1, this assumption holds for common loss functions such as square loss or binary cross entropy for several settings (e.g., kernel behavior [67]). With these two assumptions, we show that ZO-SGD has a slowdown proportional to the effective rank r, not the parameter dimension.

Lemma 3 (Global Convergence of ZO-SGD)**.** *Let $\mathcal{L}(\boldsymbol{\theta})$ be μ-PL and let there exist α such that $\operatorname{tr}(\boldsymbol{\Sigma}(\boldsymbol{\theta})) \leq \alpha(\mathcal{L}(\boldsymbol{\theta}) - \mathcal{L}^*)$ for all $\boldsymbol{\theta}$. Then after*

$$t = \mathcal{O}\left(\left(\frac{r}{n}+1\right)\cdot\underbrace{\left(\frac{\ell}{\mu}+\frac{\ell\alpha}{\mu^2 B}\right)\log\frac{\mathcal{L}(\boldsymbol{\theta}_0)-\mathcal{L}^*}{\epsilon}}_{\text{SGD rate (Lemma 4)}}\right)$$

iterations of ZO-SGD we have $\mathbb{E}[\mathcal{L}(\boldsymbol{\theta}_t)] \leq \mathcal{L}^* + \epsilon$.

5 Related work

Zeroth-order optimization Many classical lower bounds have been derived for ZO-SGD in the strongly convex and convex settings [47, 3, 79, 32, 85, 69] as well as non-convex [101]. These bounds generally depended on the number of parameters d. More recently, [100, 6, 15] showed that if the gradient has low-dimensional structure, then the query complexity scales linearly with the intrinsic dimension and logarithmically with the number of parameters, though the estimation has at least $\Omega(sd \log d)$ memory cost. Additional tricks such as sampling schedules [11] and other variance reduction methods [48, 62] can be added to ZO-SGD. ZO has inspired distributed methods [93, 43] and black-box adversarial example generation [14, 63, 17, 64] in deep learning. Ye et al. [108], Balasubramanian and Ghadimi [7] estimate the Hessian to perform ZO optimization along important directions. There are also ZO methods that optimize without estimating the gradient [38, 68, 44].

Memory-efficient backpropagation Several algorithms have been proposed to efficiently approximate backpropagation by sparsifying gradients [92, 102], approximating Jacobians [1, 19], and subsampling the computational graph [71, 2]. However, these methods may accrue large approximation errors for deep networks. Gradient checkpointing [18] reduces memory cost of backpropagation at the cost of recomputing some activations. FlashAttention [23] also reduces memory cost by recomputing attention matrices. Dettmers et al. [26, 27] explore quantization of large LMs' weights and optimizer states, which leads to memory reduction in both training and inference.

Gradient-free adaptation of large language models BBT and BBTv2 [91, 90] use evolutionary algorithms to achieve gradient-free optimization; however, due to its sensitivity to high dimensionality, BBT is limited to only optimize a low-dimension projection of prefixes and they focus on RoBERTa-large size models and few-shot settings. Other works in "black-box tuning" of LMs focus on optimizing discrete prompts without updating the model, either via reinforcement learning [16, 25, 29], ensemble [45], or iterative search [78]. Concurrent work in [106] uses iterative forward passes to improve in-context learning performance.

6 Conclusion

We have shown that MeZO can effectively optimize large LMs across many tasks and scales. Further experiments suggest that MeZO can optimize non-differentiable objectives, which backpropagation usually cannot do. Our theory illustrates why MeZO is not catastrophically slow when tuning billions of parameters. As a limitation, MeZO takes many steps in order to achieve strong performance, though we show that the per-step speedup in MeZO can often make fine-tuning with MeZO run faster than a standard implementation of fine-tuning with backpropagation. We did not explore combining MeZO with other memory-efficient methods, such as FlashAttention [23] and quantization [26], though we hope to investigate this in the future.

We are excited to explore the applicability of MeZO to a number of areas, including but not limited to: pruning, distillation, saliency, interpretability, and dataset selection for fine-tuning. Non-differentiable objectives are a particularly exciting area, given recent advances in tuning large LMs to adapt to human feedback. Conducting theoretical analyses for how these efficient gradient estimates impact the performance of different applications is also of interest.

Acknowledgements

We thank Xinyi Chen, Yin Tat Lee, Kaifeng Lyu, Tengyu Ma, Abhishek Panigrahi, Nikunj Saunshi, and Mengzhou Xia for their helpful feedback. SA and SM are funded by NSR, ONR, SRC, and Simons Foundation. JDL, AD, and EN acknowledge the support of the ARO under MURI Award W911NF-11-1-0304, the Sloan Research Fellowship, NSF CCF 2002272, NSF IIS 2107304, NSF CIF 2212262, ONR Young Investigator Award, and NSF CAREER Award 2144994. TG is supported by an IBM PhD Fellowship. This work is also partially funded by the National Science Foundation (IIS-2211779).