Revisiting Zeroth-Order Optimization for Memory-Efficient LLM Fine-Tuning: A Benchmark

Yihua Zhang[*1] **Pingzhi Li**[*2] **Junyuan Hong**[*3] **Jiaxiang Li**[*4] **Yimeng Zhang**[1] **Wenqing Zheng**[3] **Pin-Yu Chen**[5] **Jason D. Lee**[6] **Wotao Yin**[7] **Mingyi Hong**[4] **Zhangyang Wang**[3] **Sijia Liu**[1 5] **Tianlong Chen**[2 8 9]

Abstract

In the evolving landscape of natural language processing (NLP), fine-tuning pre-trained Large Language Models (LLMs) with first-order (FO) optimizers like SGD and Adam has become standard. Yet, as LLMs grow in size, the substantial memory overhead from back-propagation (BP) for FO gradient computation presents a significant challenge. Addressing this issue is crucial, especially for applications like on-device training where memory efficiency is paramount. This paper proposes a shift towards BP-free, zeroth-order (ZO) optimization as a solution for reducing memory costs during LLM fine-tuning, building on the initial concept introduced by Malladi et al. (2023). Unlike traditional ZO-SGD methods, our work expands the exploration to a wider array of ZO optimization techniques, through a comprehensive, first-of-its-kind benchmarking study across five LLM families (Roberta, OPT, LLaMA, Vicuna, Mistral), three task complexities, and five fine-tuning schemes. Our study unveils previously overlooked optimization principles, highlighting the importance of task alignment, the role of the forward gradient method, and the balance between algorithm complexity and fine-tuning performance. We further introduce novel enhancements to ZO optimization, including block-wise descent, hybrid training, and gradient sparsity. Our study offers a promising direction for achieving further memory-efficient LLM fine-tuning. Codes to reproduce all our experiments are at `https://github.com/ZO-Bench/ZO-LLM`.

[*]Equal contribution [1]Michigan State University [2]The University of North Carolina at Chapel Hill [3]UT Austin [4]University of Minnesota Twin Cities [5]IBM Research [6]Princeton University [7]DAMO Academy, Alibaba Group US [8]MIT [9]Harvard University. Correspondence to: Sijia Liu <liusiji5@msu.edu>, Tianlong Chen <tianlong@cs.unc.edu>.

Proceedings of the 41st International Conference on Machine Learning, Vienna, Austria. PMLR 235, 2024. Copyright 2024 by the author(s).

1. Introduction

Fine-tuning pre-trained large language models (LLMs) has become the *de-facto* standard in the current paradigms of natural language processing (NLP) (Raffel et al., 2023; Sanh et al., 2022). First-order (FO) optimizers, *e.g.*, SGD (Amari, 1993) and Adam (Kingma & Ba, 2014), have been the predominant choices for LLM fine-tuning. However, as LLMs continue to scale, they encounter significant memory overhead due to the back-propagation (BP) required for FO gradient computation. For example, computing the gradient of the LLM `OPT-13B` requires $12\times$ more memory cost than the model inference. This leads to the challenge of achieving *memory-efficient fine-tuning* in LLMs. Advancements in addressing this challenge could also facilitate technological breakthroughs in related areas, such as on-device training, where memory efficiency is in high demand (Han et al., 2015; Zhu et al., 2023).

To enhance memory efficiency, an emerging solution is to replace a BP-required FO optimization method with a *BP-free* optimizer during LLM fine-tuning. This was initially proposed by Malladi et al. (2023), where the FO gradient is approximated using a finite difference of function values. Despite its new application to LLM fine-tuning, the underlying optimization principle used in Malladi et al. (2023) is commonly known as *zeroth-order (ZO) optimization*, and the function value-based gradient estimate is referred to as the ZO gradient estimate (Flaxman et al., 2005; Nesterov & Spokoiny, 2017; Duchi et al., 2015; Ghadimi & Lan, 2013; Liu et al., 2020). Malladi et al. (2023) employed the classical ZO stochastic gradient descent (ZO-SGD) algorithm (Ghadimi & Lan, 2013), termed MeZO, to fine-tune the pre-trained LLMs and leveraged the BP-free characteristics of ZO optimization to reduce memory costs. However, from the perspective of ZO optimization, in addition to ZO-SGD, many other ZO optimization methods have not yet been explored in the context of LLM fine-tuning. Thus, it remains elusive whether there are potential improvements in *accuracy* and/or *efficiency* that can be achieved through a benchmarking study of ZO optimization for LLM fine-tuning. This yields the primary question to be explored:

(Q) Can we establish a benchmark for ZO optimization in LLM fine-tuning, explore the overlooked optimization principles, and advance the current state of the art?

To address (Q), our work introduces several key innovations compared to the most relevant work (Malladi et al., 2023). We explore a broader range of ZO optimization methods beyond ZO-SGD and examine various task and model types, as well as evaluation metrics. We conduct a detailed comparative analysis of different ZO optimization methods, shedding light on the often-overlooked forward gradient method (Ren et al., 2022) and other ZO optimization techniques in LLM fine-tuning. This benchmarking study helps reveal the pros and cons of these methods in accuracy and efficiency. Extended from the gained insights, we propose to further improve ZO optimization-based LLM fine-tuning using techniques of block-wise descent, hybrid ZO and FO training, and gradient sparsity. In summary, our key **contributions** are listed below.

• We create the first benchmark for ZO optimization in the context of LLM fine-tuning. This benchmark includes our investigations into 6 BP-free or ZO optimization methods, 5 LLM families, 3 tasks of varying complexities, and 5 fine-tuning schemes, covering both full-parameter and parameter-efficient fine-tuning (PEFT) approaches.

• Assisted by our benchmark, we reveal a range of previously overlooked optimization principles and insights for LLM fine-tuning with ZO optimization. These include the significance of aligning tasks to enhance ZO optimization, the role of forward gradient as an LLM fine-tuning baseline, and the trade-offs between algorithm complexity, fine-tuning accuracy, query and memory efficiency.

• In addition to a holistic assessment of existing ZO optimization methods for LLM fine-tuning, we introduce novel enhancements to ZO optimization, including block-wise ZO optimization, hybrid ZO and FO fine-tuning, and sparsity-induced ZO optimization. These proposed techniques aim to improve the accuracy of ZO LLM fine-tuning while maintaining memory efficiency.

2. Related Work

Parameter-efficient fine-tuning (PEFT). Early efforts (Houlsby et al., 2019; Lin et al., 2020) involved inserting trainable adapters, which are compact feed-forward networks, between the layers of the pre-trained model. More recently, various PEFT strategies have been proposed. For instance, *Adapter*-based methods (Houlsby et al., 2019; Chen et al., 2022; Luo et al., 2023; Karimi Mahabadi et al., 2021; Pfeiffer et al., 2020) insert a few tunable yet highly compact modules into pre-trained models. *LoRA* (Hu et al., 2021a) employs trainable low-rank weight perturbations to the pre-trained model, effectively reducing the required number of fine-tuning parameters. *Prompt-based learning* (Gao et al., 2020; Hu et al., 2021b; Tan et al., 2021) has demonstrated effectiveness in various NLP tasks. Additionally, methods like *prompt tuning* (Lester et al., 2021) and *prefix tuning* (Li & Liang, 2021) incorporate learnable continuous embeddings into the model's hidden states to condition the frozen model for specific downstream tasks. The following work (Liu et al., 2021) demonstrates its applicability on various model scales. While these state-of-the-art PEFT techniques have significantly reduced the number of parameters required for fine-tuning, they still incur memory costs associated with caching numerous activations due to the use of back-propagation (BP) (Malladi et al., 2023).

Zeroth-order optimization. Zeroth-order (ZO) optimization is a technique that uses finite differences to estimate gradients. Such algorithms utilize function value oracle only, yet share a similar algorithm structure with first-order (FO) gradient-based counterpart methods. ZO optimization usually enjoys provable (dimension-dependent) convergence guarantees, as discussed in various works (Flaxman et al., 2005; Nesterov & Spokoiny, 2017; Duchi et al., 2015; Ghadimi & Lan, 2013). These methods have drawn considerable attention due to their effectiveness in a wide range of modern machine learning (ML) challenges (Liu et al., 2020), including the adversarial attack and defense (Chen et al., 2017; Tu et al., 2019; Ye et al., 2018; Ilyas et al., 2018; Zhang et al., 2022b; Verma et al., 2023; Zhao et al., 2019; Hogan & Kailkhura, 2018; Shu et al., 2022), model-agnostic contrastive explanations (Dhurandhar et al., 2019), enhancing transfer learning through visual prompting (Tsai et al., 2020), computational graph unrolling (Vicol et al., 2023), and optimizing automated ML processes (Gu et al., 2021a; Wang et al., 2022). Beyond standard ML, it finds application in policy search in reinforcement learning (Vemula et al., 2019), network resource management (Liu et al., 2018), ML-based optimization of scientific workflows (Hoffman et al., 2022; Tsaknakis et al., 2022; Chen et al., 2024), and on-chip learning enhancements (Gu et al., 2021b).

Despite its wide range of use cases, the application of ZO optimization in ML has primarily been restricted to small model scales. This limitation is attributed to the high variance and slow convergence associated with ZO optimization, which are exacerbated by model dimensions. To scale up ZO optimization, several acceleration techniques have been proposed. These include the integration of historical data to refine ZO gradient estimators (Meier et al., 2019; Cheng et al., 2021), leveraging gradient structural information (Singhal et al., 2023) or sparsity to diminish the dependency of ZO optimization on problem size (Wang et al., 2017; Cai et al., 2022; 2021; Balasubramanian & Ghadimi, 2018; Ohta et al., 2020; Gu et al., 2021b; Chen et al., 2024), the reuse of intermediate features (Chen et al.,

2024) and random perturbation vectors (Malladi et al., 2023) in the optimization process. These advancements suggest a growing potential for the application of ZO optimization in more complex and large-scale ML problems.

BP-free training for large models. Training large models, especially LLMs, is memory-consuming due to the involved large computation graphs for BP (Ren et al., 2021; Kim et al., 2023). Thus, BP-free methods have recently become a focus in the deep learning (DL) community. Forward gradient learning (Baydin et al., 2022; Ren et al., 2022; Silver et al., 2021; Belouze, 2022), built upon the forward-mode automatic differentiation (AD), provides an alternative to ZO optimization for BP-free training. Unlike ZO optimization, it relies on the forward-mode AD to calculate a forward (directional) gradient. However, one main limitation of the forward gradient is its requirement of full access to the AD software and the deep model, making it less memory-efficient than ZO optimization and impractical for tackling black-box problems (Chen et al., 2024). The specifications of BP-free methods include greedy layer-wise learning (Nøkland & Eidnes, 2019), input-weight alignment (Boopathy & Fiete, 2022), forward-forward algorithm (Hinton, 2022), synthetic gradients (Jaderberg et al., 2017), BBT/BBTv2 evolutionary algorithms (Sun et al., 2022a;b), gradient guessing using special low dimensional structure of neural networks (Singhal et al., 2023) and other black-box methods which optimize the prompts (Prasad et al., 2023; Deng et al., 2022; Chai et al., 2022). Many of these algorithms are also motivated by seeking DL's biological interpretation.

Applying ZO optimization to fine-tune pre-trained LLMs is particularly intriguing because it combines the advantages of being BP-free and utilizing pre-training. This improves the scalability of ZO optimization to large-scale LLMs while maintaining memory efficiency. MeZO (Malladi et al., 2023) introduced a ZO-SGD algorithm to fine-tune LLMs with up to 60 billion parameters, very competitive compared to first-order optimization methods and structured fine-tuning approaches like LoRA. They also provided theoretical insights into why ZO methods can be effective for LLMs. This opens the gateway for efficient BP-free LLM fine-tuning and largely motivates our ZO benchmark study.

3. Reviewing ZO Optimization and Beyond

This work's core objective is to benchmark and harness the potential of ZO (zeroth-order) optimization in LLM finetuning, eliminating the need for (first-order) back-propagation (BP) during finetuning and thus achieving memory efficiency (Malladi et al., 2023). It is worth noting that the ZO optimization technique utilized in (Malladi et al., 2023) is primarily the basic version, specifically, ZO stochastic gradient descent (ZO-SGD). There are more advanced ZO optimization methods available, as summarized in (Liu et al., 2020). Thus, this section is dedicated to reviewing a broader range of ZO optimization approaches and shedding light on the previously overlooked principles for LLM fine-tuning.

Basics of ZO optimization. ZO optimization serves as a gradient-free alternative to first-order (FO) optimization, approximating FO gradients through function value-based gradient estimates, which we call *ZO gradient estimates*, as discussed in (Flaxman et al., 2004; Nesterov & Spokoiny, 2017; Ghadimi & Lan, 2013; Duchi et al., 2015). Thus, a ZO optimization method typically mirrors the algorithmic framework of its corresponding FO optimization counterpart. However, it substitutes the FO gradient with the ZO gradient estimate as the descent direction.

Various techniques exist for performing ZO gradient estimation. In this paper, we focus on the *randomized gradient estimator* (**RGE**) (Nesterov & Spokoiny, 2017; Duchi et al., 2015), a method that relies on the finite difference of function values along a randomly chosen direction vector. RGE has also been used by Malladi et al. (2023) to achieve memory-efficient fine-tuning for LLMs. Its preference in LLM fine-tuning is attributed to its *query efficiency*, *i.e.*, a low number of function queries. Given a scalar-valued function $f(\mathbf{x})$ where $\mathbf{x} \in \mathbb{R}^d$ of dimension d, the RGE (referred to as $\hat{\nabla} f(\mathbf{x})$) is expressed using central difference:

$$\hat{\nabla} f(\mathbf{x}) = \frac{1}{q} \sum_{i=1}^{q} \left[\frac{f(\mathbf{x} + \mu \mathbf{u}_i) - f(\mathbf{x} - \mu \mathbf{u}_i)}{2\mu} \mathbf{u}_i \right] \qquad \text{(RGE)}$$

where $\mathbf{u}_i$ is a random direction vector typically drawn from the standard Gaussian distribution $\mathcal{N}(\mathbf{0}, \mathbf{I})$, q is the number of function queries, and $\mu > 0$ is a small perturbation stepsize (also known as smoothing parameter). Malladi et al. (2023) employed RGE by setting $q = 1$ and $\mathbf{u}_i = \mathbf{u}$. Yet, it is worth noting that the query number q strikes a balance between the ZO gradient estimation variance and the query complexity. It has been shown in (Duchi et al., 2015; Liu et al., 2018) that the variance of RGE is roughly in the order of $O(d/q)$, where $O(\cdot)$ signifies the Big O notation.

The rationale behind RGE stems from the concept of the *directional derivative* (Duchi et al., 2015): As $\mu \to 0$ (letting $q = 1$), the finite difference of function values in (RGE) approaches $f'(\mathbf{x}, \mathbf{u}) := \mathbf{u}^T \nabla f(\mathbf{x})$, representing the directional derivative of $f(\mathbf{x})$ along the random direction $\mathbf{u} \sim \mathcal{N}(\mathbf{0}, \mathbf{I})$. Subsequently, RGE yields $\hat{\nabla} f(\mathbf{x}) \to f'(\mathbf{x}, \mathbf{u})\mathbf{u}$ as $\mu \to 0$. Moreover, the directional derivative provides us an unbiased gradient estimator of $\nabla f(\mathbf{x})$:

$$\mathbf{E}_{\mathbf{u}}[f'(\mathbf{x}, \mathbf{u})\mathbf{u}] = \mathbf{E}_{\mathbf{u}}[\mathbf{u}\mathbf{u}^T \nabla f(\mathbf{x})] = \nabla f(\mathbf{x}). \qquad (1)$$

With the above background, the RGE $\hat{\nabla} f(\mathbf{x})$ can be interpreted as an approximation of the FO gradient $\nabla f(\mathbf{x})$ using the directional derivative.

Forward gradient: A missing BP-free baseline in LLM fine-tuning. As a byproduct of connecting RGE to (1), we obtain the directional derivative-based gradient estimate, $\nabla f(\mathbf{x}) \approx f'(\mathbf{x}, \mathbf{u})\mathbf{u}$, which is known as the *forward gradient* (**Forward-Grad**) (Baydin et al., 2022; Ren et al., 2022). Different from RGE that relies solely on the finite difference of function values, Forward-Grad requires the use of forward mode automatic differentiation (AD) but eliminates the need for backward evaluation in the implementation of deep model fine-tuning or training. In other words, Forward-Grad is BP-free and can serve as another alternative gradient estimation method that improves the memory efficiency of LLM fine-tuning. We stress that Forward-Grad is a possibly overlooked BP-free optimizer. Given its unbiasedness as shown in (1), it could serve as an upper performance bound for ZO optimization in theory.

A focused spectrum of ZO optimization methods. Next, we provide a brief overview of the ZO optimization methods to be focused on in this work. Specifically, we will include: **ZO-SGD** (Ghadimi & Lan, 2013) that Malladi et al. (2023) has employed for LLM fine-tuning, ZO-SGD using sign-based gradient estimation (**ZO-SGD-Sign**) (Liu et al., 2019a), ZO-SGD with momentum (**ZO-SGD-MMT**) (Malladi et al., 2023), ZO-SGD with conservative gradient update (**ZO-SGD-Cons**), and the ZO variant of the Adam optimizer (**ZO-Adam**) (Chen et al., 2019).

The aforementioned methods can be unified into the following generic optimization framework in solving $\min_{\mathbf{x}} f(\mathbf{x})$:

$$\mathbf{x}_{t+1} = \mathbf{x}_t - \eta_t h(\hat{\nabla} f(\mathbf{x}_t)), \tag{2}$$

where $\mathbf{x}_t$ denotes the updated solution at the tth iteration, $\eta_t > 0$ is the learning rate, and $h(\cdot)$ is a certain descent direction post-processing operation. In (2), we omit the inclusion of the stochastic mini-batch for empirical risk minimization for ease of presentation. For instance, ZO-SGD can be expressed as (2) when $h(\hat{\nabla} f(\mathbf{x})) = \hat{\nabla} f(\mathbf{x})$. Similarly, ZO-SGD-Sign can be derived if $h(\hat{\nabla} f(\mathbf{x})) = \mathrm{sign}(\hat{\nabla} f(\mathbf{x}))$, where $\mathrm{sign}(\cdot)$ represents element-wise sign operation. Another example is ZO-SGD-Cons by setting $h(\hat{\nabla} f(\mathbf{x})) = \arg\min_{\mathbf{g} \in \{\mathbf{0}, -\hat{\nabla} f(\mathbf{x}), \hat{\nabla} f(\mathbf{x})\}} f(\mathbf{x}_t + \eta_t \mathbf{g})$. We refer readers to Appendix A for more algorithmic details of (2) as applied to the ZO optimization approaches.

Our rationale for selecting the aforementioned ZO optimization approaches for LLM fine-tuning is based on two key considerations: (1) We prioritize ZO optimization methods that require minimal modifications to the existing FO optimizer, ensuring ease of implementation for LLM fine-tuning. (2) We focus on methods with distinct algorithmic characteristics, allowing us to explore a diverse range of optimization strategies for improving LLM performance. Regarding (2), we include ZO-SGD-Sign as it employs 1-bit gradient quantization and represents one of the simplest ZO optimization methods. Additionally, we include ZO-SGD-MMT and ZO-SGD-Cons as they incorporate certain forms of 'adaptive learning' into the descent step updates. The former utilizes momentum based on historical gradient information, while the latter allows for the heuristics-based selection of the descent direction. Furthermore, ZO-Adam is one of the most complex ZO optimizers due to its utilization of moving averages and adaptive learning rates.

Table 1. Test accuracy (%) of pretrained Roberta-Large model fine-tuned on SST2 and RTE tasks using ZO and FO optimization methods with (✓) and without (✗) text alignment. The evident performance degradation is highlighted **in bold**.

Method	SST2			RTE		
	✓	✗	Difference	✓	✗	Difference
FO-SGD	91.6	91.5	0.1	70.9	61.4	9.5
ZO-SGD	89.4	79.2	**10.2**	68.7	60.4	**8.3**
ZO-Adam	89.8	79.2	**10.6**	69.2	58.7	**10.5**

Task alignment in ZO optimization for LLM fine-tuning. Scaling up ZO optimization for deep model training, as discussed in (Chen et al., 2024), is exceedingly challenging due to its high variance, which is dependent on the model size. Nevertheless, LLM pre-training offers a unique advantage by enabling the fine-tuner to start from a well-optimized pre-trained model state. This graceful model initialization makes ZO optimization potentially scalable to LLM fine-tuning tasks (Malladi et al., 2023). Even in this pretraining-finetuning paradigm, another crucial factor, which we call '**task alignment**', still plays a key role in achieving satisfactory ZO fine-tuning performance. The 'task alignment' refers to aligning the fine-tuning task with the format of the pre-training task, given by the next token or sentence prediction. For example, Gao et al. (2020); Malladi et al. (2023) have transformed downstream text classification tasks into next token prediction tasks by introducing well-crafted input prompts. These prompts serve as bridges to align the fine-tuning tasks with the pre-training ones, facilitating ZO optimization when initiated from the pre-trained model.

As a warm-up experiment, **Tab. 1** empirically justifies the importance of task alignment when applying ZO optimization to LLM fine-tuning on the simple binary classification task by comparing scenarios *with* and *without* the use of pre-defined prompts to achieve task alignment. We fine-tune the entire Roberta-Large (Liu et al., 2019b) model on SST2 (Socher et al., 2013) and RTE (Wang et al., 2019) datasets with two selected ZO methods: ZO-SGD (*i.e.*, MeZO in (Malladi et al., 2023)) and ZO-Adam. We compare their performance with that of the FO method (FO-SGD). The task alignment is achieved with the template `<CLS>SENTENCE. It was [terrible|great].<SEP>` for the SST dataset and another template `<CLS>SENTENCE1? [Yes|No], SENTENCE2.<SEP>` for RTE. As we can see, without prompt-based text alignment, there is a substantial performance drop across ZO fine-tuning methods. Both ZO-SGD and ZO-Adam yield about 10% and 8% accuracy degradation on SST2 and RTE, respectively. In contrast, FO-SGD suffers less from the absence of task alignment. This suggests that the task alignment is particularly beneficial to ZO LLM fine-tuning. It is also worth noting that crafting effective prompts for task alignment can be non-trivial, as prompt design is context-dependent and can affect the fine-tuning performance. In this work, we follow (Gao et al., 2020) and (Malladi et al., 2023) to align the fine-tuning tasks to the pretrained ones.

4. LLM Fine-Tuning Benchmarking

In this section, we delve into the empirical performance of ZO optimization in LLM fine-tuning. Our benchmarking effort includes evaluating accuracy and efficiency, accounting for different downstream task complexities (ranging from simple classification to more complicated reasoning tasks), and considering various language model types.

4.1. Benchmark Setups

LLM fine-tuning tasks, schemes, and models. We begin by introducing the tasks and the fine-tuning schemes. We focus on *three tasks*, considering their complexity from low to high, which include (1) the simplest binary classification task, Stanford Sentiment Treebank v2 (SST2) (Socher et al., 2013), (2) the question-answering task, Choice Of Plausible Alternatives (COPA) (Roemmele et al., 2011), (3) the commonsense reasoning task, WinoGrande (Sakaguchi et al., 2021), and (4) the multi-sentence reading comprehension (MultiRC) (Khashabi et al., 2018) (for efficiency evaluation only). For LLM fine-tuning on these tasks, we explore *four parameter-efficient fine-tuning (PEFT) schemes*: full-tuning (FT) that fine-tunes the entire pre-trained model, low-rank adaptation (LoRA) by imposing low-rank weight perturbations (Hu et al., 2021a), prefix-tuning (Prefix) by appending learnable parameters to token embedding (Li & Liang, 2021), and prompt-tuning (Prompt) (Liu et al., 2022) by introducing a series of learnable tokens attached to the input to adapt the fixed model to downstream tasks. We refer readers to Appx. B for details. Furthermore, we incorporate several representative language models, including Roberta-Large (Liu et al., 2019b), OPT (Zhang et al., 2022a), LLaMA2 (Touvron et al., 2023), Vicuna (Zheng et al., 2023), and Mistral (Jiang et al., 2023).

Setup and implementation details. To train the previously mentioned LLM fine-tuners, we utilize the ZO optimization methods introduced in Sec. 3. These include ZO-SGD (*i.e.* MeZO (Malladi et al., 2023)), ZO-SGD-Sign, ZO-SGD-MMT, ZO-SGD-Cons, and ZO-Adam. For comparison, we also present the performance of Forward-Grad, which relies on the forward mode auto-differentiation rather than BP. We also provide the performance of two FO optimizers: (FO-)SGD and (FO-)Adam. Before applying the aforementioned optimizers to the LLM fine-tuning tasks, we follow (Gao et al., 2020; Malladi et al., 2023) to align the fine-tuning task format with the token or sentence prediction-based pretraining task, as demonstrated in Tab. 1. We run ZO (or BP-free) optimizers and FO optimizers for 20000 and 625 iterations respectively, as outlined in (2). Note that ZO optimization takes a longer convergence time as shown in (Liu et al., 2020). When implementing (RGE), unless specified otherwise, we set the query budget per gradient estimation to $q = 1$. We determine the values of other hyperparameters, such as the smoothing parameter and learning rate, through a grid search for each method. Unless otherwise stated, following (Malladi et al., 2023), half-precision training (F16) and mixed-precision training (FP16) are adopted by default on ZO and FO methods, respectively, to reduce the memory consumption. For FP16, the model is loaded with full precision while the training is carried out in half-precision, whereas F16 means both the model and training are in float16. More implementation details can be found in Appx. C.2, and their influence on memory efficiency is discussed in Sec. 4.3.

Table 2. Performance of LLM fine-tuning on SST2 over pretrained Roberta-Large and OPT/1.3B. Best performance among ZO methods (including Forward-Grad) is highlighted in **bold**.

SST2	Roberta-Large				OPT-1.3B			
	FT	LoRA	Prefix	Prompt	FT	LoRA	Prefix	Prompt
FO-SGD	91.4	91.2	89.6	90.3	91.1	93.6	93.1	92.8
Forward-Grad	**90.1**	89.7	89.5	87.3	90.3	90.3	90.0	82.4
ZO-SGD	89.4	90.8	90.0	87.6	**90.8**	90.1	**91.4**	84.4
ZO-SGD-MMT	89.6	90.9	90.1	88.6	85.2	91.3	91.2	**86.9**
ZO-SGD-Cons	89.6	**91.6**	90.1	88.5	88.3	90.5	81.8	84.7
ZO-SGD-Sign	52.5	90.2	53.6	86.1	87.2	91.5	89.5	72.9
ZO-Adam	89.8	89.5	**90.2**	**88.8**	84.4	**92.3**	**91.4**	75.7

Evaluation metrics. We evaluate ZO LLM fine-tuning using two sets of metrics: accuracy and efficiency. Accuracy measures the fine-tuned model's test-time performance in specific tasks, such as test accuracy in classification tasks. Efficiency includes various measurements like memory efficiency (in terms of peak memory usage and GPU cost), query efficiency (*i.e.*, the number of function queries required for ZO optimization), and run-time efficiency. These metrics collectively provide insights into the resources needed for ZO LLM fine-tuning, helping assess its feasibility and cost-effectiveness in practical scenarios.

4.2. Experiment Results

ZO fine-tuning on SST2: A pilot study. In **Tab. 2**, experiments are conducted to compare the performance of different BP-free and BP-based (FO-SGD) methods on one of the simplest LLM downstream task: the binary classification task with SST2 dataset. We investigate two model architectures, the medium-sized Roberta-Large and the larger model OPT-1.3B. Several key results are summarized below.

First, ZO-Adam seems to be the most effective ZO method, achieving the best performance in 4 out of 8 fine-tuning settings. However, as will be shown later, this is achieved

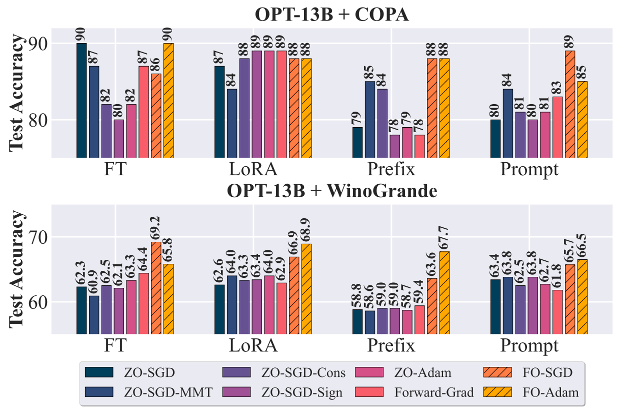

Figure 1. Results of OPT-13B on the tasks COPA and WinoGrande fine-tuned using ZO/FO optimizers in different PEFT settings.

at the cost of additional memory consumption. This is not surprising considering that ZO-Adam has the highest algorithmic complexity, as explained in Sec. 3.

Second, Forward-Grad is a competitive method compared to the ZO methods, especially in the FT (full-tuning) setting. This indicates that Forward-Grad may be suitable for problems of a larger scale, and make it a compelling baseline of ZO LLM fine-tuning. Additionally, when the complexity of the fine-tuning scheme decreases (*e.g.*, Prompt), the advantage of Forward-Grad over function value-based ZO methods diminishes.

Third, The performance of ZO methods exhibits high variance, as evidenced by the fluctuating relative rankings across different experimental scenarios, although extensive hyper-parameter search efforts have been made. For example, the effectiveness of ZO-Adam degrades dramatically in the (OPT-1.3B, Prompt) setting. In addition, the MeZO method (*i.e.*, ZO-SGD) used in (Malladi et al., 2023) does *not* always emerge as the top-performing ZO optimizer for LLM fine-tuning across various settings. This is not surprising and could be largely attributed to the high variance of RGE (Nesterov & Spokoiny, 2017; Duchi et al., 2015).

Fourth, ZO-SGD-Cons and ZO-SGD-MMT also demonstrate strong performance as ZO optimizers in LLM fine-tuning. However, ZO-SGD-Sign, the simplest ZO optimization method, tends to be the weakest approach, except in the simplest fine-tuning setting Prompt. The above observations motivate us to extend our explorations, investigating the effectiveness of ZO methods across a broader spectrum of models and more complex tasks.

ZO fine-tuning on downstream tasks COPA and WinoGrande under OPT-13B. Extended from the experiments on SST2, **Fig. 1** presents the fine-tuning performance on COPA and WinoGrande dataset using a larger model, OPT-13B. We summarize our key observations when the problem scales up and becomes more complicated.

First, compared to the previous results, the performance gap among different ZO methods is much enlarged. In the meantime, the performance gap between FO and ZO methods is also widened. For example, in the experiments with WinoGrande, the FO methods (FO-SGD and FO-Adam) outperform all the other ZO methods by a large margin. This observation shows the scalability bottleneck intrinsic to ZO methods, when dealing with larger models and/or more complex tasks.

Second, certain ZO methods exhibit exceptional stability across varied conditions: Despite a general trend towards variability, specific ZO methods, *e.g.*, ZO-Adam and ZO-SGD-MMT, demonstrate stability in their performance. This could be because these algorithms integrate variance-reduced optimization techniques (such as momentum and adaptive learning rate) to ZO optimization and become more adaptable and resilient to the variances of ZO gradient estimation (Chen et al., 2019).

Third, the LoRA tuning is consistently robust when paired with various ZO algorithms. This resilience across different ZO methods suggests that LoRA's mechanism is inherently more adaptable to various ZO optimization strategies, providing a stable and reliable tuning approach in diverse settings. We will peer into the performance of LoRA below.

In **Tab. 3**, we present how different ZO methods perform on (LoRA, COPA) and (LoRA, WinoGrande) across a wide range of LLM families. For ease of computation, we focus on a subset of ZO optimization methods, including ZO-SGD, ZO-SGD-CONS, and ZO-Adam. As we can see, in some scenarios with the COPA dataset, some BP-free methods exhibit effectiveness comparable to, or even superior to, that of FO methods (FO-SGD and FO-Adam). For example, Forward-Grad and ZO-Adam outperform the best FO method on model OPT-13B and Vicuna-7B. Conversely, for the more difficult task WinoGrande, a performance gap of $5\% \sim 6\%$ between FO and ZO methods still exists across different models.

Table 3. Performance of different LLMs finetuned with LoRA on COPA and WinoGrande using different ZO/FO methods. Table format is consistent with Tab. 2.

	OPT-13B	LLaMA2-7B	Vicuna-7B	Mistral-7B
		COPA		
FO-SGD	88	85	84	90
FO-Adam	88	84	81	90
Forward-Grad	**89**	82	**84**	88
ZO-SGD	87	**86**	83	**90**
ZO-SGD-CONS	88	85	83	89
ZO-Adam	**89**	83	**84**	89
		WinoGrande		
FO-SGD	66.9	66.9	66.5	76.4
FO-Adam	68.9	69.5	70.0	76.9
Forward-Grad	62.9	64.3	**65.6**	**70.1**
ZO-SGD	62.6	64.3	**65.6**	68.7
ZO-SGD-CONS	63.3	**64.6**	65.3	68.5
ZO-Adam	**64.0**	64.4	65.5	69.5

Table 4. The peak memory cost (in GB), the required GPU resources, and the runtime cost (in seconds) of each optimizer when fine-tuning the full OPT-13B model on MultiRC with an averaged 400 context length. The order of included optimizers is ranked based on the memory cost. The per-iteration runtime in seconds (s) is averaged over 100 iterations.

Optimizer	Memory ⇓	Consumed GPUs ⇓	Runtime Cost
ZO-SGD	**64** GB	**1**×A100	**11.1**s
ZO-SGD-Cons	**64** GB	**1**×A100	27.6s
ZO-SGD-Sign	**64** GB	**1**×A100	**11.1**s
ZO-SGD-MMT	100 GB	2×A100	**11.1**s
Forward-Grad	134 GB	2×A100	16.7s
FO-SGD	148 GB	2×A100	19.1s
ZO-Adam	158 GB	2×A100	**11.5**s
FO-Adam	245 GB	4×A100	19.4s

Efficiency analysis. In **Tab. 4**, we present a comparison of the efficiency performance of various ZO/FO optimizers when fine-tuning the full OPT-13B model on the MultiRC dataset with a batch size of 4. We evaluate the efficiency in the following dimensions: memory cost (in GB), the consumption of GPU resources (number of GPUs), and runtime cost per optimization iteration (in seconds). All the experiments are carried out in the same environment. First, from a memory efficiency standpoint, ZO-SGD, ZO-SGD-Cons, and ZO-SGD-Sign exhibit similar levels of efficiency and only necessitate a *single* GPU (A100) for LLM fine-tuning This is not surprising since these ZO methods employ relatively simple optimization steps and primarily rely on the utilization of RGE. Second, Forward-Grad seems to be the threshold beyond which the ZO optimization methods lose their memory efficiency advantage over FO methods, as seen with ZO-Adam, for example. Third, compared to FO methods, ZO optimization reduces runtime costs per iteration by $\sim 41.9\%$ for ZO-SGD versus FO-SGD.

Ablation study on query budget. Recall from (1) that Forward-Grad provides us with an unbiased gradient estimate with respect to the FO gradient, in contrast to the function value-based biased ZO gradient estimate. However, in the above experiments, we have not observed a significant advantage brought by Forward-Grad. We hypothesize that this is due to the smallest query budget $q = 1$ we used, which, despite its query efficiency, may not fully showcase the unbiased benefit. Inspired by the above, **Fig. 2** explores the impact of varying query budget (q) on the performance of Forward-Grad and ZO-SGD (*i.e.*, MeZO). As we can see, both the accuracies of Forward-Grad and ZO-SGD are improved with an increased query budget. However, the performance rise for Forward-Grad is much more evident. For example, when the query number is larger than 500, Forward-Grad outperforms ZO-SGD by a large margin $1\% \sim 2\%$ and approaches FO-SGD. These observations underscore that the inherent advantages of Forward-Grad become apparent only when a sufficient number of queries are used. The downside of using a higher query budget is a higher computation cost, which scales linearly with q.

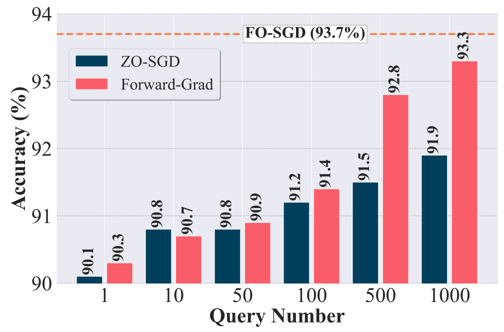

Figure 2. LoRA-based fine-tuning accuracy of OPT-1.3B on SST2 using ZO-SGD and Forward-Grad over different budgets.

4.3. An In-Depth Dissection on Memory Efficiency

In this section, we will provide provide a holistic memory profile for the ZO and FO methods, theoretically and empirically. And we will discuss how memory efficiency will be influenced by the implementation details adopted in this work, including F16 (half-precision model) and FP16 (mixed-precision training).

Theoretical analysis. The theoretical memory efficiency of all the methods with and without the memory-saving tricks are listed in **Tab. 5**. We refer readers to Appx. C.1 for a detailed analysis of these results. Several key insights can be summarized. First, compared to FO optimizers (including Forward-Grad), the memory efficiency of ZO optimizers mainly comes from two perspectives: (1) ZO methods can avoid saving the intermediate results (model states) $\mathbf{a}_l$. As we will see later, this reduction is considerable when the sequence length of the language model is large; (2) ZO methods can estimate the gradient in a layer-wise manner and thus avoid storing the gradients of the entire model. Second, the usage of FP16 in FO methods can only reduce the memory consumption of intermediate results (*i.e.*, $|\mathbf{a}|$ in Tab. 5), but not that from the model loading or optimization states' storage. In contrast, ZO methods can be easily equipped with F16, reducing memory by 50%. Third, although Forward-Grad is also a BP-free method, its memory efficiency advantage over other FO optimizers is not remarkable, mainly due to its requirement for storing the intermediate results (*i.e.*, the computational graph). Last, although ZO-Adam is memory intensive compared to ZO-SGD, it can be greatly improved by using F16 and achieves a better efficiency than FO-SGD (FP16).

Empirical results analysis. We further compare the theoretical and empirical memory costs of full fine-tuning in Tab. A1 and LoRA fine-tuning in Tab. A2. Notably, the empirical results are generally aligned with the theoretical analysis. ZO methods are generally much more efficient than their FO counterparts, and the overhead can be further reduced by loading the half-precision model (using F16).

A larger sequence length strengthens the memory bene-

Table 5. Comparison of the instant peak memory consumption of different optimizers when fine-tuning the full model. Peak memory consumption consists of the constant memory for the model (Weight Mem.), the optimizer states (Opt. State Mem.), and dynamic allocation for computing gradients and optimization (Dynamic Mem.). The cardinalities $|\mathbf{x}|$ and $|\mathbf{a}|$ correspond to the memory consumption of loading model parameters and saving the intermediate results for post-hoc backward during forward in full precision, with the subscript l representing that of a specific layer l. In Dynamic Mem., $\mathbf{x}$ is also used to denote the memory consumption for temporarily saving the gradients, as they share the same size with the corresponding parameters.

Optimizer	Weight Mem.	Dynamic Mem (Grad.&Opt.)	Opt. State Mem.
Optimizer in Full Precision			
FO-SGD	$\lvert\mathbf{x}\rvert$	$\sum_l \max\{\lvert\mathbf{a}_l\rvert, \lvert\mathbf{x}_l\rvert\}$	0
FO-Adam w/o fast foreach	$\lvert\mathbf{x}\rvert$	$\sum_l \max\{\lvert\mathbf{a}_l\rvert, \lvert\mathbf{x}_l\rvert\}$	$2\lvert\mathbf{x}\rvert$
FO-Adam	$\lvert\mathbf{x}\rvert$	$\sum_l \max\{\lvert\mathbf{a}_l\rvert, \lvert\mathbf{x}_l\rvert\}$	$3\lvert\mathbf{x}\rvert$
Forward-Grad	$\lvert\mathbf{x}\rvert$	$\lvert\mathbf{x}\rvert + \max_l \lvert\mathbf{a}_l\rvert$	0
Vanilla ZO-SGD	$\lvert\mathbf{x}\rvert$	$\lvert\mathbf{x}\rvert$	0
ZO-SGD	$\lvert\mathbf{x}\rvert$	$\max_l \lvert\mathbf{x}_l\rvert$	0
ZO-SGD MMT	$\lvert\mathbf{x}\rvert$	$\max_l \lvert\mathbf{x}_l\rvert$	$\lvert\mathbf{x}\rvert$
ZO-Adam	$\lvert\mathbf{x}\rvert$	$\max_l \lvert\mathbf{x}_l\rvert$	$2\lvert\mathbf{x}\rvert$
Optimizer in Mixed Precision Training (FP16)			
FO-SGD (**FP16**)	$\lvert\mathbf{x}\rvert$	$\max\left\{\frac{1}{2}\lvert\mathbf{a}\rvert + \frac{1}{2}\lvert\mathbf{x}\rvert, \sum_l \max\{\frac{1}{2}\lvert\mathbf{a}_l\rvert, \lvert\mathbf{x}_l\rvert\}\right\}$	0
FO-Adam (**FP16**)	$\lvert\mathbf{x}\rvert$	$\max\left\{\frac{1}{2}\lvert\mathbf{a}\rvert + \frac{1}{2}\lvert\mathbf{x}\rvert, \sum_l \max\{\frac{1}{2}\lvert\mathbf{a}_l\rvert, \lvert\mathbf{x}_l\rvert\}\right\}$	$2\lvert\mathbf{x}\rvert$
Optimizer with Half Precision Model (F16)			
ZO-SGD (**F16**)	$\frac{1}{2}\lvert\mathbf{x}\rvert$	$\max_l \frac{1}{2}\lvert\mathbf{x}_l\rvert$	0
ZO-SGD-MMT (**F16**)	$\frac{1}{2}\lvert\mathbf{x}\rvert$	$\max_l \frac{1}{2}\lvert\mathbf{x}_l\rvert$	$\frac{1}{2}\lvert\mathbf{x}\rvert$
ZO-Adam (**F16**)	$\frac{1}{2}\lvert\mathbf{x}\rvert$	$\max_l \frac{1}{2}\lvert\mathbf{x}_l\rvert$	$\lvert\mathbf{x}\rvert$

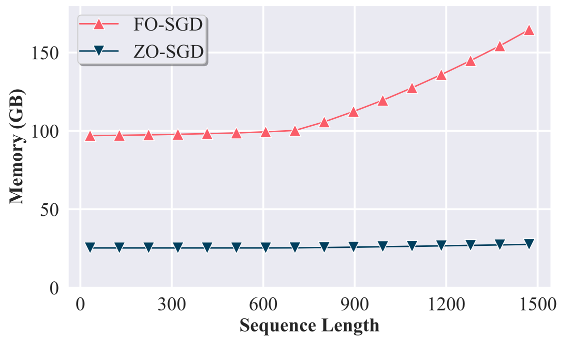

Figure 3. Peak memory comparison of full fine-tuning with FO-SGD and ZO-SGD across various sequence lengths with a fixed effective batch size of 2. Peak memory consumption was evaluated with the input of synthetic texts generated from random sequences of the specified lengths.

fits of ZO methods. We remark that the empirical results in Tab. A1 and Tab. A2 are dependent on the input sequence length. In general, a larger sequence length will directly lead to a larger activation memory consumption. To investigate its impact, we further explore how the scaling of input sequence length impacts memory usage by increasing activation memory, and the results are shown in Fig. 3. Furthermore, we examine the memory cost of LLM fine-tuning vs. the input sequence length. In **Fig. 3**, we compare the memory efficiency between ZO-SGD and FO-SGD across various sequence lengths (*i.e.* the token number per sample). As we can see, the memory consumption of ZO-SGD maintains a consistent level, since its peak memory consumption is only determined by the model parameter size (*i.e.*, the cardinality $|\mathbf{x}|$) and independent of the size of the intermediate results (*i.e.*, $|\mathbf{a}|$), see Tab. 5. In contrast, as the sequence length increases, the peak memory consumption of FO-SGD first maintains and then begins to increase. This phenomenon occurs because the peak memory consumption of FO-SGD is determined by the larger one of the per-layer activation storage and the gradient storage (*i.e.*, $\max\{|\mathbf{a}_l|, |\mathbf{x}_l|\}$). When the sequence length increases, the former begins to grow and eventually exceeds the latter at a certain point (*e.g.*, exceeding 700). Thus, ZO-SGD will exhibit a much better efficiency advantage in settings with long context lengths.

5. Extended Study to Improve ZO Fine-Tuning

Beyond the benchmarking effort in Sec. 4, we will explore algorithmic advancements to further improve the effectiveness of ZO LLM fine-tuning. We will leverage the following techniques: (1) *Block-wise ZO fine-tuning*; (2) *Hybrid ZO and FO fine-tuning*; and (3) *sparsity-induced ZO gradient estimation*. These designs aim to reduce the large variances in gradient estimation when using ZO algorithms.

Block-wise ZO optimization enhances fine-tuning performance. It has been shown in (Liu et al., 2018) that using a coordinate-wise deterministic gradient estimator can reduce ZO optimization variance, although this scheme is difficult to scale. In a similar vein, we ask if RGE when estimating a FO gradient *block-wise* can also improve the performance of ZO optimization. The key idea is to split the LLM into different blocks and then apply the ZO gradient estimator to each block of parameters. For example, OPT-1.3B entails $p = 26$ parameter blocks, necessitating p forward passes for ZO gradient estimation per fine-tuning step. Our rationale is that by conducting gradient estimation block-wise, the resulting gradient estimate's variance will be reduced, thereby

Table 6. Performance comparison of OPT-1.3B on the SST2 & WinoGrande tasks between ZO-SGD and ZO-SGD-Block. The # of parameter blocks in ZO-SGD-Block is set to $p = 26$. Thus, ZO-SGD w/ the query budget $q = 26$ has the same forward pass count as ZO-SGD-Block. MeZO corresponds to ZO-SGD w/ $q = 1$. Best performance for each task is highlighted in **bold**.

Optimizer	Forward Pass #	SST2	WinoGrande
MeZO	1	90.83	55.5
ZO-SGD ($q = 26$)	26	91.28	55.7
ZO-SGD-Block	26	**93.69**	**57.2**

potentially improving the fine-tuning performance.

In **Tab. 6**, we compare the performance of the ZO fine-tuning baseline MeZO (Malladi et al., 2023) (corresponding to ZO-SGD with a query budget of $q = 1$ in Sec. 4) with its block-wise RGE-based variant, which we term ZO-SGD-Block. For a fair comparison with ZO-SGD-Block in terms of query complexity, we also present the performance of another variant of ZO-SGD that uses the full model-wise RGE but takes the same query number q as ZO-SGD-Block per iteration. Notably, ZO-SGD-Block outperforms ZO-SGD in different query budget settings across different fine-tuning tasks, showing the benefit of *block-wise* ZO tuning.

Trade-off between performance and memory efficiency via hybrid ZO-FO training. The primary source of memory cost in LLM fine-tuning arises from BP, which involves passing gradients from the deep layers to the shallow layers of the model. To save memory, a potential approach is to confine BP within the deep layers without propagating it to the shallow layers. Moreover, ZO optimization can be employed for the shallow layers without the need for BP. The above approach of combining FO optimization for deep layers and ZO optimization for shallow layers results in a *hybrid ZO-FO fine-tuning scheme* for LLMs.

Table 7. The trade-off between memory cost (in GB) *v.s.* fine-tuning accuracy (%) using the hybrid ZO-FO training on the OPT-1.3B model over the SST2 dataset. The memory or accuracy gap vs. that of the pure ZO-SGD method is noted by Δ.

	Memory (GB)		Accuracy (%)	
ZO Layer #	Memory	ΔMemory	Accuracy	ΔAccuracy
0 (FO-SGD)	24.29	11.07	91.22	1.98
4	23.33	10.11	91.12	1.88
8	22.01	8.79	90.79	1.55
12	20.43	7.21	89.48	0.24
16	18.98	5.76	89.42	0.18
20	15.43	2.21	89.27	0.03
24 (ZO-SGD)	13.22	0.00	89.24	0.00

Tab. 7 presents the performance of using the hybrid ZO-FO fine-tuning scheme on (OPT-1.3B, SST2). We examine different variants of this hybrid scheme by deciding '*where*' to split between ZO optimization (for shallow layers) and FO optimization (for deep layers). Suppose that the model consists of n layers, and we designate the first $k \in [1, n]$ layers for ZO optimization, while the remaining $(n - k)$ layers are allocated for FO optimization. The pure ZO optimization approach corresponds to $p = n$. The results presented in Tab. 7 demonstrate that employing ZO optimization on only the first third of the model layers (*i.e.*, $k \leq 8$) can yield performance comparable to that achieved by fully utilizing FO optimization while also reducing memory usage by approximately 10%. Furthermore, when half of the layers employ ZO optimization (*i.e.*, $k \geq 12$), the performance achieved is similar to that of full ZO fine-tuning.

Table 8. Performance of fine-tuning OPT-1.3B on COPA & SST2 datasets using ZO-SGD at different sparsity ratios. A sparsity of 0% represents the baseline, the vanilla ZO-SGD, *i.e.*, MeZO (Malladi et al., 2023). Performance that surpasses the baseline (w/ 0% sparsity) is highlighted in **bold**.

COPA										
Sparsity (%)	0	10	20	30	40	50	60	70	80	90
Accuracy (%)	73.00	**75.00**	**75.00**	70.00	70.00	70.00	70.00	7.000	70.00	71.00
SST2										
Sparsity (%)	0	10	20	30	40	50	60	70	80	90
Accuracy (%)	90.83	**91.51**	**92.20**	**92.32**	**91.74**	**92.43**	**92.43**	**92.20**	**91.51**	**92.66**

Gradient pruning benefits performance. We next explore gradient pruning, a technique known for accelerating model training without compromising convergence (McDanel et al., 2022). Our key idea is to induce sparse parameter perturbations for reducing gradient estimation variance in RGE. We begin by leveraging magnitude-based pruning (Frankle & Carbin, 2018; Chen et al., 2020) to obtain the layer-wise sparsity ratios. We then generate random pruning masks (following these layer-wise sparsity ratios) and apply them to the weight perturbations in RGE per ZO fine-tuning step. **Tab. 8** shows the performance of the sparsity-induced ZO gradient estimation in LLM fine-tuning as a function of the overall sparsity ratio. It becomes evident that choosing a moderate sparsity ratio (*e.g.*, 20%) can lead to improved performance over the vanilla ZO optimizer, ZO-SGD.

6. Conclusion

This work explores the application of zeroth-order (ZO) optimization in fine-tuning LLMs. ZO optimization approximates gradients using loss differences, eliminating the need for back-propagation and activation storage. While MeZO (Malladi et al., 2023) has made strides in adapting ZO optimization for LLMs, understanding the full ZO landscape remains an open question. To address this question, we broaden the scope by considering various ZO optimization methods, task types, and evaluation metrics. We conduct the first benchmark study of different ZO optimization techniques, shedding light on their accuracy and efficiency. We also uncover the overlooked ZO optimization principles, such as task alignment and the role of forward gradient. Leveraging these insights, we propose techniques like block-wise descent, hybrid ZO and FO training, and gradient sparsity to enhance ZO optimization-based LLM fine-tuning. The proposed enhancements can further improve the fine-tuning accuracy while maintaining the memory efficiency.

Impact Statement

This paper aims to advance the optimization foundations of the memory-efficient fine-tuning of large language models (LLMs). Its potential impacts are contingent on how these fine-tuned LLMs are utilized. On the positive side, achieving memory efficiency during LLM fine-tuning could lead to significant reductions in energy consumption, contributing to the development of green AI and achieving improved performance in resource-constrained environments. However, there is a potential negative aspect in terms of misuse, as the fine-tuned models could be employed for generating misinformation, phishing attacks, or releasing copyrighted and private information. However, given the technical focus of this work, there are no specific societal consequences directly stemming from it that need to be highlighted here.

Acknowledgement

The work of Y. Zhang and S. Liu is partially supported by the U.S. Department of Energy via Lawrence Livermore National Laboratory. The work of J. D. Lee and Z. Wang is in part supported by the NSF AI Institute for Foundations of Machine Learning (IFML). The work of M. Hong and J. Li is partially supported by NSF grant CCF-1910385 and CNS-2003033. This work is also supported by an Amazon AWS gift award.

References

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

Balasubramanian, K. and Ghadimi, S. Zeroth-order (non)-convex stochastic optimization via conditional gradient and gradient updates. *Advances in Neural Information Processing Systems*, 31, 2018.

Baydin, A. G., Pearlmutter, B. A., Syme, D., Wood, F., and Torr, P. Gradients without backpropagation. *arXiv preprint arXiv:2202.08587*, 2022.

Belouze, G. Optimization without backpropagation. *arXiv preprint arXiv:2209.06302*, 2022.

Boopathy, A. and Fiete, I. How to train your wide neural network without backprop: An input-weight alignment perspective. In *International Conference on Machine Learning*, pp. 2178–2205. PMLR, 2022.

Cai, H., Lou, Y., McKenzie, D., and Yin, W. A zeroth-order block coordinate descent algorithm for huge-scale black-box optimization. *arXiv preprint arXiv:2102.10707*, 2021.

Cai, H., Mckenzie, D., Yin, W., and Zhang, Z. Zeroth-order regularized optimization (zoro): Approximately sparse gradients and adaptive sampling. *SIAM Journal on Optimization*, 32(2):687–714, 2022.

Chai, Y., Wang, S., Sun, Y., Tian, H., Wu, H., and Wang, H. Clip-tuning: Towards derivative-free prompt learning with a mixture of rewards, 2022.

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. *ICLR*, 2024.

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.

Chen, S., Ge, C., Tong, Z., Wang, J., Song, Y., Wang, J., and Luo, P. Adaptformer: Adapting vision transformers for scalable visual recognition. *Advances in Neural Information Processing Systems*, 35:16664–16678, 2022.

Chen, T., Frankle, J., Chang, S., Liu, S., Zhang, Y., Wang, Z., and Carbin, M. The lottery ticket hypothesis for pre-trained bert networks. *Advances in neural information processing systems*, 33:15834–15846, 2020.

Chen, X., Liu, S., Xu, K., Li, X., Lin, X., Hong, M., and Cox, D. Zo-adamm: Zeroth-order adaptive momentum method for black-box optimization. *NeurIPS*, 2019.

Cheng, S., Wu, G., and Zhu, J. On the convergence of prior-guided zeroth-order optimization algorithms. *Advances in Neural Information Processing Systems*, 34:14620–14631, 2021.

Deng, M., Wang, J., Hsieh, C.-P., Wang, Y., Guo, H., Shu, T., Song, M., Xing, E. P., and Hu, Z. Rlprompt: Optimizing discrete text prompts with reinforcement learning, 2022.

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.

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

Flaxman, A. D., Kalai, A. T., and McMahan, H. B. Online convex optimization in the bandit setting: gradient descent without a gradient. *arXiv preprint cs/0408007*, 2004.

Flaxman, A. D., Kalai, A. T., and McMahan, H. B. Online convex optimization in the bandit setting: Gradient descent without a gradient. In *Proceedings of the sixteenth* *annual ACM-SIAM symposium on Discrete algorithms*, pp. 385–394, 2005.

Frankle, J. and Carbin, M. The lottery ticket hypothesis: Finding sparse, trainable neural networks. *arXiv preprint arXiv:1803.03635*, 2018.

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

Ghadimi, S. and Lan, G. Stochastic first-and zeroth-order methods for nonconvex stochastic programming. *SIAM Journal on Optimization*, 23(4):2341–2368, 2013.

Gu, B., Liu, G., Zhang, Y., Geng, X., and Huang, H. Optimizing large-scale hyperparameters via automated learning algorithm. *arXiv preprint arXiv:2102.09026*, 2021a.

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, 2021b.

Han, S., Mao, H., and Dally, W. J. Deep compression: Compressing deep neural networks with pruning, trained quantization and huffman coding. *arXiv preprint arXiv:1510.00149*, 2015.

Hinton, G. The forward-forward algorithm: Some preliminary investigations. *arXiv preprint arXiv:2212.13345*, 2022.

Hoffman, S. C., Chenthamarakshan, V., Wadhawan, K., Chen, P.-Y., and Das, P. Optimizing molecules using efficient queries from property evaluations. *Nature Machine Intelligence*, 4(1):21–31, 2022.

Hogan, T. A. and Kailkhura, B. Universal decision-based black-box perturbations: Breaking security-through-obscurity defenses. *arXiv preprint arXiv:1811.03733*, 2018.

Houlsby, N., Giurgiu, A., Jastrzebski, S., Morrone, B., De Laroussilhe, Q., Gesmundo, A., Attariyan, M., and Gelly, S. Parameter-efficient transfer learning for nlp. In *International Conference on Machine Learning*, pp. 2790–2799. PMLR, 2019.

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, 2021a.

Hu, S., Ding, N., Wang, H., Liu, Z., Wang, J., Li, J., Wu, W., and Sun, M. Knowledgeable prompt-tuning: Incorporating knowledge into prompt verbalizer for text classification. *arXiv preprint arXiv:2108.02035*, 2021b.

Huang, F., Gao, S., Pei, J., and Huang, H. Accelerated zeroth-order and first-order momentum methods from mini to minimax optimization. *The Journal of Machine Learning Research*, 23(1):1616–1685, 2022.

Ilyas, A., Engstrom, L., Athalye, A., and Lin, J. Black-box adversarial attacks with limited queries and information. In *International conference on machine learning*, pp. 2137–2146. PMLR, 2018.

Jaderberg, M., Czarnecki, W. M., Osindero, S., Vinyals, O., Graves, A., Silver, D., and Kavukcuoglu, K. Decoupled neural interfaces using synthetic gradients. In *International conference on machine learning*, pp. 1627–1635. PMLR, 2017.

Jiang, A. Q., Sablayrolles, A., Mensch, A., Bamford, C., Chaplot, D. S., Casas, D. d. l., Bressand, F., Lengyel, G., Lample, G., Saulnier, L., et al. Mistral 7b. *arXiv preprint arXiv:2310.06825*, 2023.

Karimi Mahabadi, R., Henderson, J., and Ruder, S. Compacter: Efficient low-rank hypercomplex adapter layers. *Advances in Neural Information Processing Systems*, 34: 1022–1035, 2021.

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 North American Chapter of the Association for Computational Linguistics (NAACL)*, 2018.

Kim, B., Cai, H., McKenzie, D., and Yin, W. Curvature-aware derivative-free optimization. *arXiv preprint arXiv:2109.13391*, 2021.

Kim, T., Kim, H., Yu, G.-I., and Chun, B.-G. BPipe: Memory-balanced pipeline parallelism for training large language models. In Krause, A., Brunskill, E., Cho, K., Engelhardt, B., Sabato, S., and Scarlett, J. (eds.), *Proceedings of the 40th International Conference on Machine Learning*, volume 202 of *Proceedings of Machine Learning Research*, pp. 16639–16653. PMLR, 23–29 Jul 2023.

Kingma, D. P. and Ba, J. Adam: A method for stochastic optimization. *arXiv preprint arXiv:1412.6980*, 2014.

Lester, B., Al-Rfou, R., and Constant, N. The power of scale for parameter-efficient prompt tuning, 2021.

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

Lin, Z., Madotto, A., and Fung, P. Exploring versatile generative language model via parameter-efficient transfer learning, 2020.

Liu, S., Kailkhura, B., Chen, P.-Y., Ting, P., Chang, S., and Amini, L. Zeroth-order stochastic variance reduction for nonconvex optimization. volume 31, 2018.

Liu, S., Chen, P.-Y., Chen, X., and Hong, M. signSGD via zeroth-order oracle. In *International Conference on Learning Representations*, 2019a.

Liu, S., Chen, P.-Y., Kailkhura, B., Zhang, G., Hero III, A. O., and Varshney, P. K. A primer on zeroth-order optimization in signal processing and machine learning: Principals, recent advances, and applications. volume 37, pp. 43–54. IEEE, 2020.

Liu, X., Ji, K., Fu, Y., Tam, W. L., Du, Z., Yang, Z., and Tang, J. P-tuning v2: Prompt tuning can be comparable to fine-tuning universally across scales and tasks. *arXiv preprint arXiv:2110.07602*, 2021.

Liu, X., Ji, K., Fu, Y., Tam, W., Du, Z., Yang, Z., and Tang, J. P-tuning: Prompt tuning can be comparable to fine-tuning across scales and tasks. In *Proceedings of the 60th Annual Meeting of the Association for Computational Linguistics (Volume 2: Short Papers)*, pp. 61–68, 2022.

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. *arXiv preprint arXiv:1907.11692*, 2019b.

Luo, G., Huang, M., Zhou, Y., Sun, X., Jiang, G., Wang, Z., and Ji, R. Towards efficient visual adaption via structural re-parameterization. *arXiv preprint arXiv:2302.08106*, 2023.

Malladi, S., Gao, T., Nichani, E., Damian, A., Lee, J. D., Chen, D., and Arora, S. Fine-tuning language models with just forward passes. *arXiv preprint arXiv:2305.17333*, 2023.

McDanel, B., Dinh, H., and Magallanes, J. Accelerating dnn training with structured data gradient pruning. In *2022 26th International Conference on Pattern Recognition (ICPR)*, pp. 2293–2299. IEEE, 2022.

Meier, F., Mujika, A., Gauy, M. M., and Steger, A. Improving gradient estimation in evolutionary strategies with past descent directions. *arXiv preprint arXiv:1910.05268*, 2019.

Nesterov, Y. and Spokoiny, V. Random gradient-free minimization of convex functions. *Foundations of Computational Mathematics*, 17:527–566, 2017.

Nøkland, A. and Eidnes, L. H. Training neural networks with local error signals. In *International conference on machine learning*, pp. 4839–4850. PMLR, 2019.

Ohta, M., Berger, N., Sokolov, A., and Riezler, S. Sparse perturbations for improved convergence in stochastic zeroth-order optimization. In *Machine Learning, Optimization, and Data Science: 6th International Conference, LOD 2020, Siena, Italy, July 19–23, 2020, Revised Selected Papers, Part II 6*, pp. 39–64. Springer, 2020.

Pfeiffer, J., Rücklé, A., Poth, C., Kamath, A., Vulić, I., Ruder, S., Cho, K., and Gurevych, I. Adapterhub: A framework for adapting transformers. *arXiv preprint arXiv:2007.07779*, 2020.

Prasad, A., Hase, P., Zhou, X., and Bansal, M. Grips: Gradient-free, edit-based instruction search for prompting large language models, 2023.

Raffel, C., Shazeer, N., Roberts, A., Lee, K., Narang, S., Matena, M., Zhou, Y., Li, W., and Liu, P. J. Exploring the limits of transfer learning with a unified text-to-text transformer, 2023.

Ren, J., Rajbhandari, S., Aminabadi, R. Y., Ruwase, O., Yang, S., Zhang, M., Li, D., and He, Y. Zero-offload: Democratizing billion-scale model training, 2021.

Ren, M., Kornblith, S., Liao, R., and Hinton, G. Scaling forward gradient with local losses. *arXiv preprint arXiv:2210.03310*, 2022.

Roemmele, M., Bejan, C. A., and Gordon, A. S. Choice of plausible alternatives: An evaluation of commonsense causal reasoning. In *2011 AAAI Spring Symposium Series*, 2011.

Sakaguchi, K., Bras, R. L., Bhagavatula, C., and Choi, Y. Winogrande: An adversarial winograd schema challenge at scale. *Communications of the ACM*, 64(9):99–106, 2021.

Sanh, V., Webson, A., Raffel, C., Bach, S. H., Sutawika, L., Alyafeai, Z., Chaffin, A., Stiegler, A., Scao, T. L., Raja, A., Dey, M., Bari, M. S., Xu, C., Thakker, U., Sharma, S. S., Szczechla, E., Kim, T., Chhablani, G., Nayak, N., Datta, D., Chang, J., Jiang, M. T.-J., Wang, H., Manica, M., Shen, S., Yong, Z. X., Pandey, H., Bawden, R., Wang, T., Neeraj, T., Rozen, J., Sharma, A., Santilli, A., Fevry, T., Fries, J. A., Teehan, R., Bers, T., Biderman, S., Gao, L., Wolf, T., and Rush, A. M. Multitask prompted training enables zero-shot task generalization, 2022.

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*, 2022.

Silver, D., Goyal, A., Danihelka, I., Hessel, M., and van Hasselt, H. Learning by directional gradient descent. In *International Conference on Learning Representations*, 2021.

Singhal, U., Cheung, B., Chandra, K., Ragan-Kelley, J., Tenenbaum, J. B., Poggio, T. A., and Yu, S. X. How to guess a gradient. *arXiv preprint arXiv:2312.04709*, 2023.

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.

Sun, T., He, Z., Qian, H., Zhou, Y., Huang, X., and Qiu, X. Bbtv2: Towards a gradient-free future with large language models, 2022a.

Sun, T., Shao, Y., Qian, H., Huang, X., and Qiu, X. Black-box tuning for language-model-as-a-service. In *International Conference on Machine Learning*, pp. 20841–20855. PMLR, 2022b.

Tan, Z., Zhang, X., Wang, S., and Liu, Y. Msp: Multi-stage prompting for making pre-trained language models better translators. *arXiv preprint arXiv:2110.06609*, 2021.

Touvron, H., Martin, L., Stone, K., Albert, P., Almahairi, A., Babaei, Y., Bashlykov, N., Batra, S., Bhargava, P., Bhosale, S., et al. Llama 2: Open foundation and fine-tuned chat models. *arXiv preprint arXiv:2307.09288*, 2023.

Tsai, Y.-Y., Chen, P.-Y., and Ho, T.-Y. Transfer learning without knowing: Reprogramming black-box machine learning models with scarce data and limited resources. In *International Conference on Machine Learning*, pp. 9614–9624. PMLR, 2020.

Tsaknakis, I., Kailkhura, B., Liu, S., Loveland, D., Diffenderfer, J., Hiszpanski, A. M., and Hong, M. Zeroth-order sciml: Non-intrusive integration of scientific software with deep learning. *arXiv preprint arXiv:2206.02785*, 2022.

Tu, C.-C., Ting, P., Chen, P.-Y., Liu, S., Zhang, H., Yi, J., Hsieh, C.-J., and Cheng, S.-M. Autozoom: Autoencoder-based zeroth order optimization method for attacking black-box neural networks. In *Proceedings of the AAAI Conference on Artificial Intelligence*, pp. 742–749, 2019.

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

Vicol, P., Kolter, Z., and Swersky, K. Low-variance gradient estimation in unrolled computation graphs with es-single. *arXiv preprint arXiv:2304.11153*, 2023.

Wang, A., Singh, A., Michael, J., Hill, F., Levy, O., and Bowman, S. R. GLUE: A multi-task benchmark and analysis platform for natural language understanding. 2019. In the Proceedings of ICLR.

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.

Wang, Y., Du, S., Balakrishnan, S., and Singh, A. Stochastic zeroth-order optimization in high dimensions. *arXiv preprint arXiv:1710.10551*, 2017.

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.

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*, 2022a.

Zhang, Y., Yao, Y., Jia, J., Yi, J., Hong, M., Chang, S., and Liu, S. How to robustify black-box ml models? a zeroth-order optimization perspective. *ICLR*, 2022b.

Zhao, P., Liu, S., Chen, P.-Y., Hoang, N., Xu, K., Kailkhura, B., and Lin, X. On the design of black-box adversarial examples by leveraging gradient-free optimization and operator splitting method. In *Proceedings of the IEEE/CVF International Conference on Computer Vision*, pp. 121–130, 2019.

Zheng, L., Chiang, W.-L., Sheng, Y., Zhuang, S., Wu, Z., Zhuang, Y., Lin, Z., Li, Z., Li, D., Xing, E., et al. Judging llm-as-a-judge with mt-bench and chatbot arena. *arXiv preprint arXiv:2306.05685*, 2023.

Zhu, S., Voigt, T., Ko, J., and Rahimian, F. On-device training: A first overview on existing systems, 2023.

Appendix

A. Zeroth-Order Optimization Algorithms

Zeroth-order optimization addresses the minimization or maximization of an objective function $f : \mathbb{R}^n \to \mathbb{R}$ without the use of derivatives:

$$\min_{\mathbf{x}\in\mathbb{R}^n} f(\mathbf{x})$$

These methods are pivotal when the function is non-differentiable, gradient computation is expensive, or the function evaluations are noisy. Random gradient estimation (RGE) provides a surrogate for gradients in zeroth-order optimization by sampling function evaluations. The gradient $\nabla f(\mathbf{x})$ at a point $\mathbf{x}$ can be approximated as:

$$\hat{\nabla} f(\mathbf{x}) := \frac{f(\mathbf{x}+\mu\mathbf{u}) - f(\mathbf{x}-\mu\mathbf{u})}{2\mu} \cdot \mathbf{u}$$

where $\mathbf{u}$ is a standard Gaussian vector, and μ is a small scalar. This estimation facilitates the use of gradient-based methods solely based on function evaluations. Utilizing this gradient estimator, we summarize the existing zeroth-other algorithms as follows

▷ *ZO-SGD.* (Ghadimi & Lan, 2013) The method directly update the parameters by the estimated gradient:

$$\mathbf{x}_{t+1} = \mathbf{x}_t - \eta_t \hat{\nabla} f(\mathbf{x}_t).$$

▷ *ZO-Sign-SGD.* (Liu et al., 2019a) The intuition of ZO-Sign-SGD is to make the gradient estimation more robust to noise since the sign operation could mitigate the negative effect of (coordinate-wise) gradient noise of large variance. ZO-Sign-SGD uses two-side RGE and update the parameters as follows,

$$\mathbf{x}_{t+1} = \mathbf{x}_t - \eta_t \operatorname{sign}\Big(\frac{f(\mathbf{x}+\mu\mathbf{u}) - f(\mathbf{x}-\mu\mathbf{u})}{2\mu}\Big)\mathbf{u}$$

where u is a standard Gaussian vector.

▷ *ZO-SGD with Momentum.* (Huang et al., 2022) The stochastic gradient estimated from a batch may suffer from a large variance. Momentum uses a moving average to estimate the global gradient and update the parameters by

$$\mathbf{x}_{t+1} = \mathbf{x}_t - \eta_t \mathbf{m}_t,$$

with the momentum defined as

$$\mathbf{m}_t = \beta_t \mathbf{m}_{t-1} + \hat{\nabla} f(\mathbf{x}_t).$$

▷ *ZO-SGD with Conservative Gradient Update.* (Kim et al., 2021) This method is adapted from (Kim et al., 2021) to update the parameter in a conservative way: we pick up the point that corresponds to the smallest loss value. The update writes:

$$\mathbf{x}_{t+1} = \operatorname*{arg\,min}_{\mathbf{y}\in\{\mathbf{x}_t, \mathbf{x}_t - \eta_t\hat{\nabla} f(\mathbf{x}_t), \mathbf{x}_t + \eta_t\hat{\nabla} f(\mathbf{x}_t)\}} f(\mathbf{y})$$

Due to the large variance introduced by the zeroth-order RGE estimator, we believe that the conservative update could correct the wrong step made by ZO-SGD, yielding better convergence results.

▷ *ZO-Adam.* (Chen et al., 2019) Similar to the ZO-SGD with momentum, ZO-Adam uses momentum to estimate the gradients. In addition, ZO-Adam adaptively penalizes the learning rate by a diagonal matrix $\mathbf{V}_t$ to reduce the noise. In summary, the parameters will be updated by

$$\mathbf{x}_{t+1} = \mathbf{x}_t - \eta_t \mathbf{V}_t^{-1/2}\mathbf{m}_t,$$

where the momentum $\mathbf{m}_t$, second raw momentum estimate $\mathbf{v}$ and normalization matrix $\mathbf{V}$ are defined as

$$\begin{aligned}
\mathbf{m}_t &= \beta_{1,t}\mathbf{m}_{t-1} + (1-\beta_{1,t})\hat{\nabla} f(\mathbf{x}_t), \\
\mathbf{v}_t &= \beta_2 \mathbf{v}_{t-1} + (1-\beta_2)[\hat{\nabla} f(\mathbf{x}_t)]^2, \\
\mathbf{V}_t &= \mathrm{Diag}(\max(\mathbf{v}_t, \mathbf{v}_{t-1})).
\end{aligned}$$

▷ *Forward Gradient.*(Baydin et al., 2022) Forward Gradient was proposed to get unbiased stochastic estimation of directional derivatives. The update is formulated as

$$\mathbf{x}_{t+1} = \mathbf{x}_t - \eta_t(\nabla f(\mathbf{x}_t)^\top \mathbf{u})\mathbf{u}$$

, where $\mathbf{u}$ is the standard Gaussian random vector. Though the Forward Gradient uses the first-order gradient $\nabla f(\mathbf{x}_t)$, it uses the Jacobian-Vector Product (JVP) during the forward pass of training to reduce the computation and memory consumption. Therefore, the memory complexity is still comparably low compared to back-propagation-based approaches.

B. Preliminaries of Parameter-Efficient Fine-Tuning (PEFT)

In our benchmark, we consider three PEFT methods, including {LoRA, prefix tuning, prompt tuning}.
1) *Low-Rank Adaptation (LoRA).* LoRA modifies a pre-trained model by introducing trainable low-rank matrices, enabling fine-tuning with a limited parameter budget. Given a weight matrix $\mathbf{W} \in \mathbb{R}^{m\times n}$ in a transformer model, LoRA decomposes it as:

$$\mathbf{W}' = \mathbf{W} + \mathbf{BA} \tag{A1}$$

where W is the original weight matrix, $\mathbf{B} \in \mathbb{R}^{m\times r}$ and $\mathbf{A} \in \mathbb{R}^{r\times n}$ are the low-rank matrices, and $r \ll \min(m, n)$ represents the rank. During fine-tuning, only $\mathbf{B}$ and $\mathbf{A}$ are updated, keeping $\mathbf{W}$ frozen.

2*) Prompt Tuning.* Prompt tuning introduces a series of trainable tokens, or prompts, to guide the model's predictions. Let $\mathbf{x}$ be the input sequence and $\mathbf{P} \in \mathbb{R}^{l\times d}$ the matrix representing the prompt embeddings, where l is the length of the prompt and d is the embedding dimension. The model input is then:

$$\hat{\mathbf{x}} = [\mathbf{P}; \mathbf{E}(\mathbf{x})] \tag{A2}$$

where $\mathbf{E}(\mathbf{x})$ is the embedding of the original input $\mathbf{x}$, and $\hat{\mathbf{x}}$ represents the concatenated input of the prompt and the original input. During fine-tuning, only the prompt embeddings $\mathbf{P}$ are learned, with the rest of the model parameters kept frozen.

3*) Prefix Tuning.* Prefix tuning extends the idea of prompt tuning to the attention mechanism of transformer models. Given an input sequence $\mathbf{x}$, the model processes it with additional context vectors $\mathbf{C}_k$ and $\mathbf{C}_v$ serving as keys and values in the attention mechanism:

$$\text{Attention}(\mathbf{Q}, \mathbf{K}, \mathbf{V}) = \text{softmax}\left(\frac{\mathbf{Q}(\mathbf{K}+\mathbf{C}_k)^{\mathrm{T}}}{\sqrt{d_k}}\right)(\mathbf{V}+\mathbf{C}_v) \tag{A3}$$

where $\mathbf{Q}$, $\mathbf{K}$ and $\mathbf{V}$ represent the query, key, and value matrices in the attention mechanism, $\mathbf{C}_k \in \mathbb{R}^{l\times d_k}$, $\mathbf{C}_v \in \mathbb{R}^{l\times d_v}$, and l is the length of the prefix. During training, only $\mathbf{C}_k$ and $\mathbf{C}_v$ are updated, and the original model parameters are frozen.

Algorithm A1 A General Pipeline for A FO/ZO Optimizer

```
Input: Model with forward function $f$, parameters $\mathbf{x}$, previous optimizer state $\mathbf{s}_{\text{opt}}$
Memory Overview: $\mathbf{s}$: States of optimizer/model, $\boldsymbol{\tau}$: Temporary variables (e.g., for copy/paste)
Step 0: Model Loading
• Function: Initialize the model with parameter $\mathbf{x}$;
• Memory Consumption: $\mathbf{x}$;
• Memory Release: N/A.
Step 1: Forward Pass
• Function: Compute loss $\ell(x)$, save forward pass states $\mathbf{s}_{\text{fwd}}$, and use temporary variables $\boldsymbol{\tau}_{\text{fwd}}$.
• Memory Consumption: $\ell(x), \mathbf{s}_{\text{fwd}}, \boldsymbol{\tau}_{\text{fwd}}$;
• Memory Release: Release $\boldsymbol{\tau}_{\text{fwd}}$ after computation.
Step 2: Backward Pass
• Function: Calculate gradients w.r.t. $\mathbf{x}$, generate backward states $\mathbf{s}_{\text{bwd}}$, employ temporary variables $\boldsymbol{\tau}_{\text{bwd}}$;
• Memory Consumption: $\mathbf{s}_{\text{bwd}}, \boldsymbol{\tau}_{\text{bwd}}$;
• Memory Release: Release $\mathbf{s}_{\text{fwd}}, \boldsymbol{\tau}_{\text{bwd}}$ after gradients are computed.
Step 3: Optimization Step
• Function: Update $\mathbf{x}$ and $\mathbf{s}_{\text{opt}}$ using gradients, utilize temporary variables $\boldsymbol{\tau}_{\text{opt}}$.
• Memory Consumption: $\mathbf{x}, \mathbf{s}_{\text{opt}}, \boldsymbol{\tau}_{\text{opt}}$.
• Memory Release: Release $\mathbf{s}_{\text{bwd}}, \boldsymbol{\tau}_{\text{opt}}$ post-update.
Output: Loss $\ell$, updated parameters $\mathbf{x}$, and optimizer state $\mathbf{s}_{\text{opt}}$.
```

C. How to Implement Memory-Efficient ZO/FO Optimizers?

The memory efficiency of an optimizer heavily depends on its implementation details. In this section, we will discuss these implementation details and provide an holistic memory profile of all the optimizers discussed above. We remark that the discussions in this section are based on the PyTorch framework. In general, the memory efficiency of an optimizer is defined by the peak memories consumed during training in a specific setting, which primarily depends on the maximum number of variables stored at a certain time point. To dissect the memory consumption of different optimizers, we summarized a general model parameter updating pipeline in Algorithm A1, which involves four main steps. First, the program needs to load the model with the full parameter $\mathbf{x}$ with either full or half-precision. This is an inevitble consumption for all optimizers. Second, a forward pass will yield a loss value and store any involved intermediate state of the model $\mathbf{s}_{\text{fwd}}$. This process may also requires storing any other temporary variables $\boldsymbol{\tau}_{\text{fwd}}$ (*e.g.* an extra copy in float16 of model weight in FO-SGD FP16), which will be immediately released after the forward is complete. Third, the loss is back-propagated in a backward mode utilizing the stored state $\mathbf{s}_{\text{fwd}}$. Similarly, this process involves storing the backward states $\mathbf{s}_{\text{bwd}}$ and temporarily storing $\boldsymbol{\tau}_{\text{bwd}}$. After the gradients and state $\mathbf{s}_{\text{bwd}}$ are computed, the memory cost from $\mathbf{s}_{\text{fwd}}$ will be immediately released. Last, with the computed gradient and states, the parameters and optimizer state will be updated, and the memory of $\mathbf{s}_{\text{bwd}}$ will be released. Typically, $\mathbf{s}_{\text{bwd}}$ will be the stored gradients. Since the forward states and gradients are computed layer-wise, the memory consumption will be sequentially transmitted from the state to the gradient. According to the life-cycle, the memory consumption will be summarized as two types: *constant memory* that exists through the training life-cycle, and *dynamic allocation* that only exists in one iteration mainly for gradient computing. Though dynamic allocation only temporarily exist, it can increase the peak memory. As a result, we summarize the peak memory consumption as

$$\underbrace{|\mathbf{x}| + |\mathbf{s}_{\text{opt}}|}_{\text{constant}} + \underbrace{\max\{|\mathbf{s}_{\text{fwd}}| + |\boldsymbol{\tau}_{\text{fwd}}|, |\mathbf{s}_{\text{bwd}}| + |\boldsymbol{\tau}_{\text{bwd}}|, |\mathbf{s}_{\text{bwd}}| + |\boldsymbol{\tau}_{\text{opt}}|\}}_{\text{dynamic allocation}}.$$

If the forward and backward are computed per layer and the temporary states can be voided, then the term $\max\{|\mathbf{s}_{\text{fwd}}| + |\boldsymbol{\tau}_{\text{fwd}}|, |\mathbf{s}_{\text{bwd}}| + |\boldsymbol{\tau}_{\text{bwd}}|\}$ will be replaced by $\sum_l \max\{|\mathbf{s}_{\text{fwd},l}|, |\mathbf{s}_{\text{bwd},l}|\}$, where l represents any layer number, leading to a memory consumption as shown below:

$$|\mathbf{x}| + \max\left\{\sum_l \max\{|\mathbf{s}_{\text{fwd},l}|, |\mathbf{s}_{\text{bwd},l}|\}, |\mathbf{s}_{\text{bwd}}| + |\boldsymbol{\tau}_{\text{opt}}|\right\} + |\mathbf{s}_{\text{opt}}|.$$

In the next, we will go through all the FO/ZO optimizers considered in this work and discuss their memory efficiency one by one.

C.1. Theoretical Memory Efficiency Analysis of Different Optimizers

FO-SGD. Following Algorithm A1, the stored model/optimizer states are defined as

$$\begin{aligned} \mathbf{s}_{\text{fwd}} &= \oplus_{l=1}^{L} \mathbf{a}_l \\ \mathbf{s}_{\text{bwd}} &= \oplus_{l=1}^{L} \mathbf{g}_l \\ \mathbf{s}_{\text{opt}} &= \emptyset, \end{aligned}$$

where $\mathbf{a}_l$ represents the total activations being stored for computing the backward gradients $\mathbf{g}_l$, and $\oplus$ denotes the vector concatenation operation. Therefore, the total memory consumption is

$$\begin{aligned} &|\mathbf{x}| + \max\left\{\sum_l \max\{|\mathbf{s}_{\text{fwd},l}|, |\mathbf{s}_{\text{bwd},l}|\}, |\mathbf{s}_{\text{bwd}}|\right\} + |\mathbf{s}_{\text{opt}}| \\ &= |\mathbf{x}| + \max\left\{\sum_l \max\{|\mathbf{a}_l|, |\mathbf{g}_l|\}, |\mathbf{g}|\right\} + 0 \\ &= |\mathbf{x}| + \sum_l \max\{|\mathbf{a}_l|, |\mathbf{g}_l|\} \end{aligned}$$

FO-SGD FP16. We use $\bar{\mathbf{a}}$ and $\bar{\mathbf{x}}$ to denote the 16-bit version of $\mathbf{a}$ and $\mathbf{x}$ (half-precision), respectively. The states are defined as:

$$\begin{aligned} \mathbf{s}_{\text{fwd}} &= \oplus_{l=1}^{L} (\bar{\mathbf{a}}_l \oplus \bar{\mathbf{x}}_l) \\ \mathbf{s}_{\text{bwd}} &= \oplus_{l=1}^{L} \mathbf{g}_l \\ \mathbf{s}_{\text{opt}} &= \emptyset. \end{aligned}$$

Here, besides the activations $\bar{\mathbf{a}}_l$ stored in float16 for backward computing, there exists an extra copy of model weight in float16 in mixed precision training. This extra copy of model weight is a temporary memory, $|\tau_{\text{fwd}}| = |\bar{\mathbf{x}}|$, during forward computing. Therefore, the total memory consumption is

$$\begin{aligned} &|\mathbf{x}| + \max\{|\mathbf{s}_{\text{fwd}}| + |\boldsymbol{\tau}_{\text{fwd}}|, |\mathbf{s}_{\text{bwd}}| + |\boldsymbol{\tau}_{\text{bwd}}|, |\mathbf{s}_{\text{bwd}}|\} + |\mathbf{s}_{\text{opt}}| \\ &= |\mathbf{x}| + \max\left\{|\bar{\mathbf{a}}| + |\bar{\mathbf{x}}|, \sum_l \max\{|\bar{\mathbf{a}}_l|, |\mathbf{g}_l|\}\right\}. \end{aligned}$$

Vanilla ZO-SGD. For the vanilla implementation of ZO-SGD, the forward states include the random vector $\mathbf{z}$ and projected gradient δ, i.e.,

$$\mathbf{s}_{\text{fwd}} = \mathbf{z} \oplus \delta$$

The projected gradient is computed by differentiating two forward passes, where the random vectors are given by:

$$\mathbf{z} = [\mathbf{z}_l \sim \mathcal{N}]_{l=1}^{L},$$

which have the same dimension as the model parameters $\mathbf{x}$. Then, the backward state is the gradients estimated by $\mathbf{z}$ and no optimizer state is involved, i.e.,

$$\begin{aligned} \mathbf{s}_{\text{bwd}} &= \mathbf{g} \\ \mathbf{s}_{\text{opt}} &= \emptyset. \end{aligned}$$

Therefore, the total memory consumption is:

$$\begin{aligned} &|\mathbf{x}| + \max\left\{\sum_l \max\{|\mathbf{s}_{\text{fwd},l}|, |\mathbf{s}_{\text{bwd},l}|\}, |\mathbf{s}_{\text{bwd}}|\right\} + |\mathbf{s}_{\text{opt}}| \\ &= |\mathbf{x}| + \max\{|\mathbf{x}| + |\mathbf{g}|\} + 0 \\ &= |\mathbf{x}| + |\mathbf{g}|. \end{aligned}$$

ZO-SGD w/ memory reduction trick (ZO-SGD). In this work, the ZO-SGD method is by default implemented with the random state trick outlined in MeZO (Malladi et al., 2023) to reduce the memory consumption. The key idea of this trick is to save the random seed used to generate the random vectors instead of directly saving the random vectors themselves. By manually set the random seed, the same corresponding random vectors for each layer $\mathbf{z}_l$ can be generated *on demand* and used for perturbation without any additional storage incurred:

$$\texttt{rng} = \text{randomState}(\mathcal{S}), \ \mathbf{z} = [\mathbf{z}_l \sim \mathcal{N}_{\text{rng}}]_{l=1}^{L}$$

where `rng` is a pseudo-random variable generator. Therefore, all the states needed are:

$$\begin{aligned} \mathbf{s}_{\text{fwd}} &= \oplus_{l=1}^{L} \mathbf{x}_l \\ \mathbf{s}_{\text{bwd}} &= \oplus_{l=1}^{L} \mathbf{g}_l \\ \mathbf{s}_{\text{opt}} &= \emptyset, \end{aligned}$$

and this yields the total memory consumption:

$$\begin{aligned} &|\mathbf{x}| + \max\left\{\sum_l \max\{|\mathbf{s}_{\text{fwd},l}|, |\mathbf{s}_{\text{bwd},l}|\}, |\mathbf{s}_{\text{bwd}}|\right\} + |\mathbf{s}_{\text{opt}}| \\ &= |\mathbf{x}| + \max\{\mathbf{g}_l\} \end{aligned}$$

For simplicity, we use the name ZO-SGD to denote ZO-SGD w/ memory reduction trick throughout the paper.

ZO-SGD-Momentum. ZO-SGD with Momentum is similar to ZO-SGD, but it consumes extra memory for momentum storage, which shares the same size as the model parameter. This is considered as an optimizer state in Algorithm A1. Therefore, the states are defined as:

$$\begin{aligned} \mathbf{s}_{\text{fwd}} &= \oplus_{l=1}^{L} \mathbf{x}_l \\ \mathbf{s}_{\text{bwd}} &= \oplus_{l=1}^{L} \mathbf{g}_l \\ \mathbf{s}_{\text{opt}} &= \oplus_{l=1}^{L} \mathbf{x}_l. \end{aligned}$$

Similar to ZO-SGD, the total memory consumption is:

$$\begin{aligned} &|\mathbf{x}| + \max\left\{\sum_l \max\{|\mathbf{s}_{\text{fwd},l}|, |\mathbf{s}_{\text{bwd},l}|\}, |\mathbf{s}_{\text{bwd}}|\right\} + |\mathbf{s}_{\text{opt}}| \\ &= |\mathbf{x}| + \max\{\mathbf{g}_l\} + |\mathbf{x}| \\ &= 2|\mathbf{x}| + \max\{\mathbf{g}_l\}. \end{aligned}$$

Forward Gradient. For FG, the forward states include the random vector $\mathbf{z}$ and projected gradient δ. The projected gradient is computed by the Jacobian-Vector Product (JVP) and forward gradient, *i.e.*, $\frac{\partial^\top f}{\partial \mathbf{x}}\mathbf{z}$, where the random vector is:

$$\mathbf{z} = [\mathbf{z}_l \sim \mathcal{N}]_{l=1}^{L}.$$

In addition to $\mathbf{z}$ and δ, JVP itself needs to store intermediate results. For example, consider a simplified two-layer network $\mathbf{y} = f_2(f_1(\mathbf{x}))$. Suppose the output of the first layer is $\mathbf{a} = f_1(\mathbf{x})$. The corresponding JVP can be formulated as

$$\frac{\partial \mathbf{y}}{\partial \mathbf{x}}^\top \mathbf{v} = \frac{\partial \mathbf{y}}{\partial \mathbf{a}}^\top \left(\frac{\partial \mathbf{a}}{\partial \mathbf{x}}^\top \mathbf{v} \right),$$

where $\left(\frac{\partial \mathbf{a}}{\partial \mathbf{x}}^\top \mathbf{v}\right)$ is computed in the first layer and needs to take the memory of size $|\mathbf{a}|$. Now, we can summarize the memory required for the forward state $\mathbf{s}_{\text{fwd}}$ as

$$\mathbf{s}_{\text{fwd}} = \mathbf{z} \oplus \mathbf{a} \oplus \delta.$$

Then, the backward state is the gradients estimated by $\mathbf{z}$, *i.e.*,

$$\mathbf{s}_{\text{bwd}} = \oplus_{l=1}^{L} (\delta \mathbf{z}_l)$$
$$\mathbf{s}_{\text{opt}} = \emptyset.$$

Therefore, the total memory consumption is

$$|\mathbf{x}| + \max \left\{ \sum_l \max\{|\mathbf{s}_{\text{fwd},l}|, |\mathbf{s}_{\text{bwd},l}|\}, |\mathbf{s}_{\text{bwd}}| \right\} + |\mathbf{s}_{\text{opt}}|$$
$$= |\mathbf{x}| + \max\{\mathbf{g} + \max_l \mathbf{a}_l, \mathbf{g}\} + 0$$
$$= |\mathbf{x}| + |\mathbf{g}| + \max_l\{\mathbf{a}_l\}.$$

Forward Gradient (FG) without State is not supported in PyTorch. This is because PyTorch requires the $\mathbf{z}$ to be pre-computed, thus the $\mathbf{s}_{\text{fwd}}$ cannot be reduced to stateless implementation. Therefore, there is no implementation if we are using the forward gradient provided by PyTorch forward-mode automatic differentiation (forward-mode AD).

Though not possible with the PyTorch built-in Automatic Differentiation tools, FG without State is still possible. Specifically, for each layer, we keep the order when the $\mathbf{z}_l$ is computed, then the $\mathbf{z}_l$ can be computed on demand. The pursuit of a more memory-efficient FG computation in practice remains an open question, reserved for future work.

C.2. Other Implementation Details

Half precision (F16). Most modern LLMs are served at 16-bit float precision. By default, models will be loaded at full precision, i.e., 32-bit. To speed up inference and improve memory efficiency, models can be loaded at 16-bit precision, namely *F16*. We do F16 for ZO methods that do not require differentiation. Note for Forward Gradient, the F16 cannot be loaded for auto differentiation. For FO methods, the half-precision is not allowed for the same reason.

Mixed precision (FP16). For FO methods, mixed precision is a common practice to speed up gradient computation and reduce memory complexity. We denote the mixed precision (i.e., 16-bit on computing gradients) as FP16. Note that FP16 will not reduce the memory consumption for storing models or gradients but only affect the gradient computation and intermediate results.

For FO-SGD (FP16), the memory consumption for computing gradient is

$$\max\{\frac{1}{2}|\mathbf{a}| + \frac{1}{2}|\mathbf{x}|, \sum_l \max\{\frac{1}{2}|\mathbf{a}_l|, |\mathbf{x}_l|\}\},$$

where $\frac{1}{2}|\mathbf{a}| + \frac{1}{2}|\mathbf{x}|$ is the memory for model and activations in 16 bit and $\max\{\frac{1}{2}|\mathbf{a}_l|, |\mathbf{x}_l|\}$ is the memory of activation and full-precision gradient.

The 'foreach' implementation of Adam. The PyTorch implementation of Adam will use the so-called 'foreach' implementation to speed up the computation. At a high level, the foreach implementation will merge all the layers' weight into one tensor during Adam updates. Though foreach can speed up computation, it will demand extra memory to store all the weights.

Table A1. Memory consumption comparison for **full fine-tuning** (FT) evaluated on OPT-13B model with the MultiRC dataset (400 tokens per example on average). Notations are consistent with Tab. 5. The theoretical and empirical values below can be mutually corroborated, where $|\mathbf{x}| \approx 48$ GB, $\sum_l \max\{|\mathbf{a}_l|, |\mathbf{x}_l|\} \approx 49$ GB. Note $|\mathbf{a}|$ are dependent on the sequence length of the input.

Optimizer	Theoretical Mem.	Empirical Mem.
FO-SGD	$\sum_l \max\{\|\mathbf{a}_l\|, \|\mathbf{x}_l\|\} + \|\mathbf{x}\|$	97 GB
FO-Adam w/o fast foreach	$\sum_l \max\{\|\mathbf{a}_l\|, \|\mathbf{x}_l\|\} + 3\|\mathbf{x}\|$	195 GB
FO-Adam	$\sum_l \max\{\|\mathbf{a}_l\|, \|\mathbf{x}_l\|\} + 4\|\mathbf{x}\|$	239 GB
Forward Grad	$2\|\mathbf{x}\| + \max_l \|\mathbf{a}_l\|$	103 GB
Vanilla ZO-SGD	$2\|\mathbf{x}\|$	96 GB
ZO-SGD	$\max_l \|\mathbf{x}_l\| + \|\mathbf{x}\|$	51 GB
ZO-SGD MMT	$\max_l \|\mathbf{x}_l\| + 2\|\mathbf{x}\|$	100 GB
ZO-Adam	$\max_l \|\mathbf{x}_l\| + 3\|\mathbf{x}\|$	151 GB
FO-SGD (**FP16**)	$\max\left\{\frac{1}{2}\|\mathbf{a}\| + \frac{1}{2}\|\mathbf{x}\|, \sum_l \max\{\frac{1}{2}\|\mathbf{a}_l\|, \|\mathbf{x}_l\|\}\right\} + \|\mathbf{x}\|$	98 GB
FO-Adam (**FP16**)	$\max\left\{\frac{1}{2}\|\mathbf{a}\| + \frac{1}{2}\|\mathbf{x}\|, \sum_l \max\{\frac{1}{2}\|\mathbf{a}_l\|, \|\mathbf{x}_l\|\}\right\} + 4\|\mathbf{x}\|$	239 GB
ZO-SGD (**F16**)	$\max_l \frac{1}{2}\|\mathbf{x}_l\| + \frac{1}{2}\|\mathbf{x}\|$	25 GB
ZO-SGD-MMT (**F16**)	$\max_l \frac{1}{2}\|\mathbf{x}_l\| + \|\mathbf{x}\|$	49 GB
ZO-Adam (**F16**)	$\max_l \frac{1}{2}\|\mathbf{x}_l\| + \frac{3}{2}\|\mathbf{x}\|$	74 GB

C.3. Additional Experiments

We compare the empirical memory costs of full fine-tuning in **Tab. A1** and LoRA fine-tuning in **Tab. A2**. Notably, the empirical results are generally aligned with the theoretical analysis. ZO methods are generally much more efficient

Table A2. Memory consumption comparison for **LoRA** fine-tuning evaluated on OPT-13B model with the MultiRC dataset (400 tokens per example on average). Other settings are consistent with Tab. A1. The memory consumption of the LoRA adapter's parameters is assumed to be significantly smaller than $|\mathbf{x}|$ and $|\mathbf{a}|$, and thus is omitted in this table.

Optimizer	Theoretical Mem.	Empirical Mem.
FO-SGD	$\sum_l \max\{|\mathbf{a}_l|, |\epsilon_l|\} + |\mathbf{x}|$	69 GB
FO-Adam w/o fast foreach	$\sum_l \max\{|\mathbf{a}_l|, |\epsilon_l|\} + 2|\epsilon| + |\mathbf{x}|$	69 GB
FO-Adam	$\sum_l \max\{|\mathbf{a}_l|, |\epsilon_l|\} + 3|\epsilon| + |\mathbf{x}|$	69 GB
Forward Grad	$|\epsilon| + |\mathbf{x}| + \max_l |\mathbf{a}_l|$	55 GB
Vanilla ZO-SGD	$|\mathbf{x}| + |\epsilon|$	52 GB
ZO-SGD	$\max_l |\epsilon_l| + |\mathbf{x}|$	52 GB
ZO-SGD MMT	$\max_l |\epsilon_l| + |\epsilon| + |\mathbf{x}|$	52 GB
ZO-Adam	$\max_l |\epsilon_l| + 2|\epsilon| + |\mathbf{x}|$	52 GB
FO-SGD (**FP16**)	$\max\left\{\frac{1}{2}|\mathbf{a}| + \frac{1}{2}|\mathbf{x}|, \sum_l \max\{\frac{1}{2}|\mathbf{a}_l|, |\epsilon_l|\}\right\} + |\mathbf{x}|$	92 GB
FO-Adam (**FP16**)	$\max\left\{\frac{1}{2}|\mathbf{a}| + \frac{1}{2}|\mathbf{x}|, \sum_l \max\{\frac{1}{2}|\mathbf{a}_l|, |\epsilon_l|\}\right\} + 3|\epsilon| + |\mathbf{x}|$	93 GB
ZO-SGD (**F16**)	$\max_l \frac{1}{2}|\epsilon_l| + \frac{1}{2}|\mathbf{x}|$	25 GB
ZO-SGD-MMT (**F16**)	$\max_l \frac{1}{2}|\epsilon_l| + \frac{1}{2}|\epsilon| + \frac{1}{2}|\mathbf{x}|$	25 GB
ZO-Adam (**F16**)	$\max_l \frac{1}{2}|\epsilon_l| + |\epsilon| + \frac{1}{2}|\mathbf{x}|$	25 GB

than their FO counterparts, and the overhead can be further reduced by loading the half-precision model (using F16).