Zeroth-Order Fine-Tuning of LLMs with Extreme Sparsity

Wentao Guo [1 2] **Jikai Long** [3] **Yimeng Zeng** [4] **Zirui Liu** [5] **Xinyu Yang** [6] **Yide Ran** [3]
Jacob R. Gardner [4] **Osbert Bastani** [4] **Christopher De Sa** [2] **Xiaodong Yu** [3] **Beidi Chen** [6] **Zhaozhuo Xu** [3]

Abstract

Zeroth-order optimization (ZO) is a memory-efficient strategy for fine-tuning Large Language Models using only forward passes. However, the application of ZO fine-tuning in memory-constrained settings such as mobile phones and laptops is still challenging since full precision forward passes are infeasible. In this study, we address this limitation by integrating sparsity and quantization into ZO fine-tuning of LLMs. Specifically, we investigate the feasibility of fine-tuning an extremely small subset of LLM parameters using ZO. This approach allows the majority of un-tuned parameters to be quantized to accommodate the constraints of limited device memory. Our findings reveal that the pre-training process can identify a set of "sensitive parameters" that can guide the ZO fine-tuning of LLMs on downstream tasks. Our results demonstrate that fine-tuning 0.1% sensitive parameters in the LLM with ZO can outperform the full ZO fine-tuning performance, while offering wall-clock time speedup. Additionally, we show that ZO fine-tuning targeting these 0.1% sensitive parameters, combined with 4 bit quantization, enables efficient ZO fine-tuning of an Llama2-7B model on a GPU device with less than 8GiB of memory and notably reduced latency.

1. Introduction

Large language models (LLMs) have demonstrated superior performance in general-purpose language generation (Brown et al., 2020; Radford et al., 2019; Liu et al., 2019). Despite their success, it remains necessary to fine-tune LLMs for specific tasks to achieve optimal results. However, fine-tuning LLMs often requires much more memory resulting from 4 parts: **(1)** the weight parameter itself; **(2)** the optimizer state, which contains the information about the past gradient (Kingma & Ba, 2015); **(3)** the weight gradient used to update the parameters; **(4)** the activation cached to calculate the weight gradient (Liu et al., 2024b); In previous work like QLoRA (Dettmers et al., 2023), it can reduce both **(1)** and **(2)** by combining weight quantization and low-rank adaption (Hu et al., 2021), which enables fine-tuning huge LLMs under data-center level GPUs. However, under more memory-constraint hardware like cell phones, the memory of caching **(3)** and **(4)** still cannot be overlooked. This limits the adaptability of LLMs, especially when personalizing them for edge devices.

Efficient ZO LLM Fine-Tuning with Sparsity. Although ZO methods remove the need for backpropagation, a significant drawback of these methods is the slow convergence rate(Zhao et al., 2024; Liu et al., 2024a). A recent approach addresses this by fine-tuning with a sparse mask (Liu et al., 2024a; Zhang et al., 2024), achieving approximately $\sim 75\%$ sparsity. Nonetheless, this sparsity level barely reduces computational overhead, as the latency during the forward pass with *even* $\sim 90\%$ sparsity is still comparable to that of dense matrix operations. [1] This latency increase can greatly impact user experience on applications such as personal assistants, where even a twofold increase in latency is perceptible. In addition, merging the sparse weights back into the base model is impractical on these devices due to memory constraints prohibiting dequantization and re-quantization. Empirical evidence suggests that higher sparsity levels can significantly decrease the time required for sparse matrix operations. This raises the question:

Is it possible to leverage the benefits of higher sparsity levels in reducing inference latency while preserving performance on downstream tasks? If so, how far can sparsity be pushed in this context?

Our Proposal: ZO LLM Fine-Tuning with Fisher-Informed, Transferable Sparsity. In this paper, we answer the raised research question by proposing an efficient sparse ZO LLM fine-tuning strategy. We observe an extreme spar-

[1]Princeton University [2]Cornell University [3]Stevens Institute of Technology [4]University of Pennsylvania [5]Rice University [6]Carnegie Mellon University. Correspondence to: Wentao Guo <wg0420@princeton.edu>, Zhaozhuo Xu <zxu79@stevens.edu>.

Proceedings of the 2^{nd} *Efficient Systems for Foundation Models Workshop at the International Conference on Machine Learning (ICML)*, Vienna, Austria. PMLR 235, 2024. Copyright 2024 by the author(s).

[1]We show this in Figure 6b in Appendix D.4.2.

sity pattern in LLM parameters: a subset, determined by selecting the top k magnitude entries from the empirical Fisher information matrix, is effective for ZO fine-tuning. Moreover, we find this sparsity pattern can be obtained through LLM's continuous pre-training process and be transferred to various downstream tasks without modification.

Summary of Contributions:

- We identify that only an extremely small portion (**0.1%**) of LLM parameters should be updated during ZO LLM fine-tuning. Moreover, we utilize this insight to guide the memory-efficient on-device personalization of LLMs by low-bit quantization of model parameters.
- We observe the sparsity pattern in LLM pre-training can be transferred across different tasks while maintaining good ZO performance. Based on this observation, we develop a computational framework to perform parameter-efficient ZO fine-tuning of LLMs.
- We conduct extensive experiments across various LLMs and demonstrate that our method achieves competitive performance across various downstream tasks.

2. ZO optimization

Given a dataset $\mathcal{D} = \{(\mathbf{x}_1, y_1), \ldots, (\mathbf{x}_n, y_n)\}$ and a loss function f with model parameters $\mathbf{w} \in \mathbb{R}^d$, ZO optimizer estimates the gradient via ZO surrogate gradient estimator.

Definition 2.1 (**Simultaneous Perturbation Stochastic Approximation (SPSA)** (Spall, 1992)). SPSA estimates the gradient with a data example $(\mathbf{x}, y)$, a small constant $\epsilon \in \mathbb{R}$, and a sampled random vector $\mathbf{z} \sim \mathcal{N}(\mathbf{0}, \mathbf{I}_d)$ as follows:

$$\hat{g}(\mathbf{w}, (\mathbf{x}, y), \mathbf{z}) = \frac{f(\mathbf{w} + \epsilon\mathbf{z}; (\mathbf{x}, y)) - f(\mathbf{w} - \epsilon\mathbf{z}; (\mathbf{x}, y))}{2\epsilon}\mathbf{z} \tag{1}$$

Definition 2.2 (**ZO-SGD update rule**). ZO-SGD uses ZO surrogate gradients to update parameters $\mathbf{w}_t$ with learning rate η_t and a data example $(\mathbf{x}_t, y_t)$ at timestep t:

$$\mathbf{w}_{t+1} = \mathbf{w}_t - \eta_t \hat{g}_{\mathbf{w}}(\mathbf{w}_t, (\mathbf{x}_t, y_t), \mathbf{z}_t) \tag{2}$$

3. Chasing Extreme Sparsity in ZO LLM Fine-Tuning

In this section, we describe the extreme sparsity pattern we observed in LLMs and how we utilize it for efficient ZO fine-tuning including on-device personalization of LLMs.

3.1. Extreme Sparsity Pattern in LLM

Sensitive parameters are *parameters whose corresponding FO coordinate-wise gradient square values are maximized.*

Definition 3.1 (**Sensitive parameter mask**). Given model parameters $\mathbf{w}$, a loss function f, a data example $(\mathbf{x}, y)$, a sensitive sparse mask $\mathbf{m}_k \in \{0, 1\}^d$ with k nonzero entries

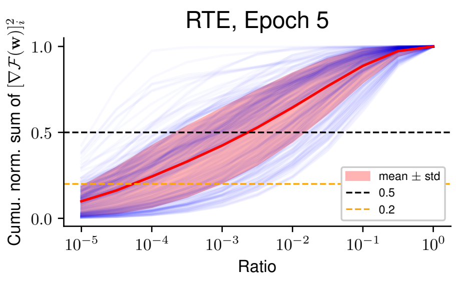


Figure 1: Cumulative normalized sum of coordinate-wise gradient square $[\nabla \mathcal{F}(\mathbf{w})]_i^2$ of linear layers during Llama2-7B (Touvron et al., 2023) fine-tuning. For each linear layer, we first sort parameters by the decreasing order of their gradient square value $[\nabla \mathcal{F}(\mathbf{w})]_i^2, i \in [d_{\text{layer}}]$, and we take the cumulative sum and normalize it to draw a blue curve, and the red-shaded region is the mean ± std of all blue curves. More similar figures are in Figure 4. **We observe that roughly 0.1% parameters in all linear layers contribute about 50% gradient norm square.**

($\sum_i \mathbf{m}(i) = k$) is defined as

$$\mathbf{m}_k = \text{argmax}_{\mathbf{m}} \|\mathbf{m} \odot \nabla f(\mathbf{w}; (\mathbf{x}, y))\|_2^2. \tag{3}$$

In the context of ZO optimization, we will update sensitive parameters *only*. Denote that $\bar{\mathbf{z}} = \mathbf{z} \odot \mathbf{m}_k$. We will modify the SPSA gradient estimator from $\hat{g}(\mathbf{w}, (\mathbf{x}, y), \mathbf{z})$ to $\hat{g}(\mathbf{w}, (\mathbf{x}, y), \bar{\mathbf{z}})$, and accordingly:

Definition 3.2 (**Sensitive sparse ZO-SGD update rule**).

$$\mathbf{w}_{t+1} = \mathbf{w}_t - \eta_t \hat{g}_{\mathbf{w}}(\mathbf{w}_t, (\mathbf{x}_t, y_t), \bar{\mathbf{z}}_t) \tag{4}$$

The theoretical intuition of our method is described in Appendix B.1 and a convergence rate under standard non-convex optimization setting is provided in Appendix B.2.

3.2. An Opportunity for On-Device LLM Personalization

As LLMs are often pre-trained with user-agnostic public datasets, personalizing LLMs with individual user's preferences and needs before real-world deployment is vital (Tan et al., 2024b; Mairittha et al., 2020). Transferring the user-specific data to upstream cloud before fine-tuning LLMs would raise privacy concerns (Xu et al., 2018), while performing full fine-tuning on personal devices would easily exceed the device memory budget. We utilize the following observations to make on-device LLM fine-tuning possible.

The essence of our method is to *extract fixed surrogate sensitive parameters with gradients from pre-training datasets and only optimize them during ZO fine-tuning*.

Transferability of Sparsity Pattern in ZO Fine-Tuning Recent works show fine-tuning performance of LLMs could

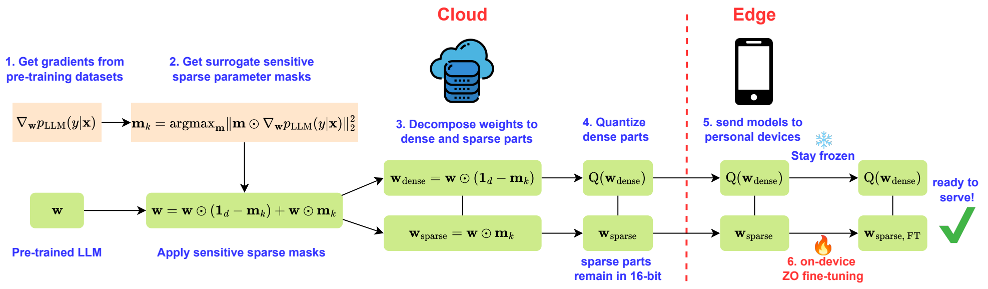

Cloud
Edge
1. Get gradients from pre-training datasets
2. Get surrogate sensitive sparse parameter masks
$\nabla_{\mathbf{w}} p_{\text{LLM}}(y|\mathbf{x})$
$\mathbf{m}_k = \text{argmax}_{\mathbf{m}} \|\mathbf{m} \odot \nabla_{\mathbf{w}} p_{\text{LLM}}(y|\mathbf{x})\|_2^2$
3. Decompose weights to dense and sparse parts
4. Quantize dense parts
5. send models to personal devices
Stay frozen
$\mathbf{w}$
$\mathbf{w} = \mathbf{w} \odot (\mathbf{1}_d - \mathbf{m}_k) + \mathbf{w} \odot \mathbf{m}_k$
$\mathbf{w}_{\text{dense}} = \mathbf{w} \odot (\mathbf{1}_d - \mathbf{m}_k)$
$\mathbf{w}_{\text{sparse}} = \mathbf{w} \odot \mathbf{m}_k$
$\text{Q}(\mathbf{w}_{\text{dense}})$
$\mathbf{w}_{\text{sparse}}$
$\mathbf{w}_{\text{sparse, FT}}$
ready to serve!
Pre-trained LLM
Apply sensitive sparse masks
sparse parts remain in 16-bit
6. on-device ZO fine-tuning

Figure 2: On-device LLM personalization workflow via integrating sensitive sparse ZO optimization with quantization.

Table 1: Performance of difference methods on Llama2-7B fine-tuning tasks. In the first column, "Q" means the full model is quantized with 4-bit quantization method (SqueezeLLM (Kim et al., 2023)), and "ZO" means the model is finetuned with ZO-SGD optimizer. For each cell, we use the same hyperparameters and repeat it with 3 random seeds. We report the average and standard deviation of test set accuracy in the format of **mean$_{\text{std}}$**. In last 2 columns, "Acc" means the average test set accuracy and "Rank" means the average rank among all methods across tasks.

	Methods	SST-2	RTE	CB	BoolQ	WSC	WiC	COPA	Acc	Rank
Q, ZO	**Sensitive (C4, static)**	**94.7**$_{0.4}$	**74.7**$_{1.2}$	**66.7**$_{2.2}$	**83.0**$_{0.5}$	57.4$_{3.9}$	**65.2**$_{0.9}$	85.0$_{2.2}$	**75.2**	**2.43**
	LoRA	93.8$_{0.6}$	64.7$_{1.1}$	64.9$_{4.7}$	79.7$_{1.1}$	**61.5**$_{2.1}$	59.8$_{0.1}$	**85.7**$_{0.5}$	72.9	4.29
	Prefix	80.5$_{4.3}$	65.5$_{1.2}$	63.1$_{3.0}$	80.3$_{0.2}$	54.5$_{11.4}$	58.3$_{1.3}$	82.0$_{0.8}$	69.2	5.86
ZO	**Sensitive (task, static)**	**94.8**$_{0.1}$	**73.6**$_{0.9}$	**69.1**$_{2.2}$	**83.5**$_{0.8}$	57.4$_{4.7}$	64.2$_{1.1}$	**83.7**$_{2.4}$	**75.2**	**2.29**
	Random (static)	94.1$_{0.3}$	68.0$_{1.7}$	64.9$_{3.4}$	77.0$_{0.7}$	**59.6**$_{3.6}$	**64.8**$_{1.1}$	83.3$_{1.7}$	73.1	4.14
	Full fine-tuning	94.6$_{0.5}$	73.3$_{5.1}$	66.7$_{0.8}$	81.9$_{0.8}$	58.0$_{4.3}$	61.9$_{0.2}$	82.7$_{1.7}$	74.2	3.57
	Zero-shot	89.0$_{0.0}$	57.8$_{0.0}$	32.1$_{0.0}$	69.9$_{0.2}$	50.2$_{0.0}$	36.5$_{0.0}$	79.0$_{0.0}$	59.2	7.29
	ICL	94.8$_{0.2}$	71.5$_{4.3}$	72.6$_{15.2}$	77.5$_{4.6}$	53.2$_{1.1}$	61.1$_{4.3}$	87.0$_{2.2}$	74.0	3.43

be attributed to a *fixed* sparse subset of parameters (Panigrahi et al., 2023; Malladi et al., 2023b). The similarity of gradient features during fine-tuning would imply that we do not need to re-select our sensitive parameters during fine-tuning i.e. select once *before fine-tuning* should be sufficient. In our method, we would need $\nabla_{\mathbf{w}} f(\mathbf{w}_{\text{after FT}}; (\mathbf{x}, y)) \sim \nabla_{\mathbf{w}} f(\mathbf{w}_{\text{before FT}}; (\mathbf{x}, y))$ to hold. This is supported by Malladi et al. (2023b)'s claim that some LLM fine-tuning tasks would demonstrate fixed (gradient) features, and we also empirically observe it in Figure 5 in Appendix D.3.

Zeroth-order parameter-efficient optimization on fixed sparse parameters The sparse optimization on *fixed* parameters can be implemented as a parameter-efficient optimization workflow, which will reduce the perturbation and updating time during ZO optimization. Suppose we have derived a sensitive sparse mask $\mathbf{m}_k$, and we know it is fixed during fine-tuning. Instead of applying $\mathbf{m}_k$ to $\mathbf{z}$, we would apply it directly to $\mathbf{w}$ and extract the nonzero parts as below:

$$\mathbf{w}_{\text{sparse}} = \mathbf{w} \odot \mathbf{m}_k, \quad \mathbf{w}_{\text{dense}} = \mathbf{w} \odot (\mathbf{1}_d - \mathbf{m}_k) \tag{5}$$

Denote $\mathbf{z}_{k,t} \sim \mathcal{N}(\mathbf{0}_k, \mathbf{I}_k)$ as the Gaussian perturbation sampled in timestep t. We will determine $\mathbf{w}_{\text{sparse}}$ before fine-tuning and optimize on $\mathbf{w}_{\text{sparse}}$ *only* and leave $\mathbf{w}_{\text{dense}}$ frozen during fine-tuning. In this case, our sensitive sparse ZO-SGD update rule will become:

$$\mathbf{w}_{\text{sparse},t+1} = \mathbf{w}_{\text{sparse},t} - \eta_t \hat{g}(\mathbf{w}_{\text{sparse},t}, (\mathbf{x}_t, y_t), \mathbf{z}_{k,t}) \tag{6}$$

Integration with quantization. Since we know that we can obtain surrogate sensitive sparse masks before fine-tuning. We would first decompose sensitive $\mathbf{w}$ to $\mathbf{w}_{\text{sparse}}$ and $\mathbf{w}_{\text{dense}}$. We would then quantize $\mathbf{w}_{\text{dense}}$ and *only* optimize $\mathbf{w}_{\text{sparse}}$. During this process, we use surrogate gradient information that many post-training quantization (PTQ) algorithms already have: they need gradients from pre-training datasets to calibrate quantization errors.

On-device personalization workflow. The workflow is illustrated in Figure 2. The high-level overview is that we use surrogate gradient information from pre-training datasets $\nabla_{\mathbf{w}} p_{\text{LLM}}(y|\mathbf{x})$ to extract sensitive parameters $\mathbf{w}_{\text{sparse}}$ and keep $\mathbf{w}_{\text{sparse}}$ in 16 bits, while we quantize the remaining dense weights $\mathbf{w}_{\text{dense}}$ (Step 1-4). We send $\mathbf{w}_{\text{sparse}}$ and $Q(\mathbf{w}_{\text{dense}})$ to personal devices (Step 5), and **we perform on-device ZO fine-tuning only on $\mathbf{w}_{\text{sparse}}$** (Step 6). In this case, our memory consumption is *nearly minimum*: we can fine-tune a Llama-2 7B model under 8 GiB GPU memory *without any offloading*. This would satisfy the memory constraint by a wide range of edge or mobile devices as illustrated in Table 3 in Appendix D.1.

4. Experiments

4.1. On-device personlization

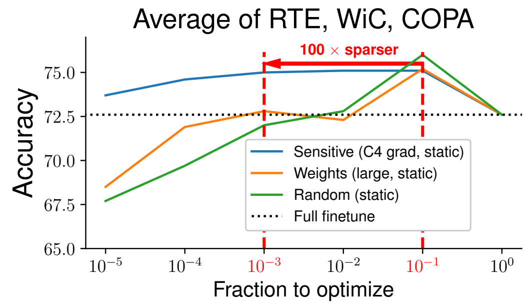
We validate whether our sensitive sparse ZO optimization method would fit with on-device personalization pipeline described in Section 3.2 with Table 1. Results for Mistral-7B and OPT-6.7B are in Table 4 in Appendix D.4.1. We follow the exact recipe as described Figure 2 to report a number as "Sensitive (C4, static)", where we only optimize 0.1% sensitive parameters on top of a 4-bit quantized model. As ZO fine-tuning happens *after* model is quantized, ablating on extracting 0.1% random subsets of parameters would produce a *different* quantized model. So we choose to report the result for optimizing a fixed random subset on top of the 16-bit model as the "Random (static)".

We also compare with optimizing with LoRA (Hu et al., 2021) and Prefix Tuning (Li & Liang, 2021) with ZO-SGD optimizer on top of the *same* quantized model. We follow the LoRA r and α and prefix length shown in Malladi et al. (2023a), and for LoRA, we add it to all linear layers same as where our sensitive parameters are extracted. We find that integrating sensitive sparse ZO optimization with on-device personalization pipelines would still yield good performance exceeding all baselines across models and tasks. Particularly, the performance is higher than In Context Learning (ICL), and ZO full fine-tuning in 16 bits. In addition, we have surpassed other ZO-PEFT methods and random sparse ZO fine-tuning methods. This demonstrates the superiority of optimizing sensitive parameters *only* in ZO fine-tuning recipes. We also notice that optimizing sensitive parameters derived from C4 gradients still produce close results as from task-specific gradients. This indicates optimizing *surrogate* sensitive parameters is still empirically successful.

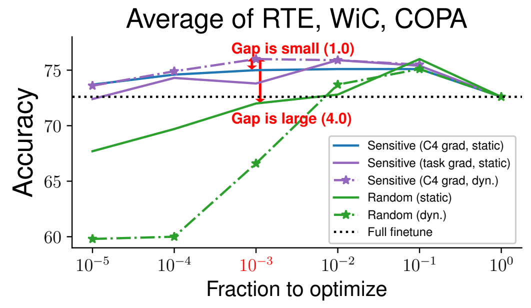
4.2. Ablation study: Effectiveness of Sparse ZO Fine-Tuning on Sensitive Parameters

We investigate the performance of optimizing our sensitive parameters versus other subsets of parameters. Our baseline sparsity methods are random subsets and weight outliers. As illustrated in Figure 3a, we can find that ZO fine-tuning would benefit from sparse optimization, as all methods would achieve higher than ZO full fine-tuning at 90% sparsity. However, only sensitive parameters would maintain its performance as we move to the extreme sparsity region ($> 99\%$). In fact, *the performance curve of sensitive parameters w.r.t. different sparsity levels is near a flat curve*, which indicates the performance loss by moving from 90% to 99.9% is minimal. Therefore, we can optimize **100 $\times$ less parameters** compared with random and weight outliers and still get same performance.

We also validate whether optimizing *fixed* and *surrogate* sensitive parameters should still yield satisfactory performance. In Figure 3b, we compare the performance of optimizing sensitive parameters with C4 gradients with its theoretical upper bound: *fixed* sensitive parameters derived from task-specific gradients as the solid line and its dynamic version as the dash-dotted line. We also include the fixed and dynamic random subset parameters as a baseline. We can find that the gap of sensitive parameters between deriving from C4 gradients and task-specific gradients at sparsity level 99.9% is *small* and blue line is still far above the random subset and full-finetuning baseline.



(a) Optimizing sensitive parameters with C4 gradients versus optimizing weights with largest magnitude (outliers) and random subsets of weights. The trainable parameters are all determined before fine-tuning and other parameters are kept unchanged.



(b) Optimizing sensitive parameters with C4 gradients versus task-specific gradients. "Static" means the parameters to optimize are determined before fine-tuning and other parameters are kept unchanged during fine-tuning. "Dyn." means the parameters to optimize will change every 100 training steps based on each algorithm.

Figure 3: Average performance of optimizing sensitive parameters in Llama2-7B fine-tuning on RTE, WiC, and COPA tasks. The complete version of subfigure 3a and 3b can be found in subfigure 7a and 7b in Appendix D.4.3.

5. Conclusion

We show that sensitive parameters provided by the pretraining process can effectively assist in ZO LLMs fine-tuning. Our experiment results also demonstrate that the quantization of parameters other than sensitive parameters allows us to effectively perform ZO fine-tuning of LLMs on limited memory devices.

References

Brown, T., Mann, B., Ryder, N., Subbiah, M., Kaplan, J. D., Dhariwal, P., Neelakantan, A., Shyam, P., Sastry, G., Askell, A., et al. Language models are few-shot learners. *Advances in neural information processing systems*, 33: 1877–1901, 2020.

Dao, T. Flashattention-2: Faster attention with better parallelism and work partitioning. *arXiv preprint arXiv:2307.08691*, 2023.

Dettmers, T., Pagnoni, A., Holtzman, A., and Zettlemoyer, L. Qlora: Efficient finetuning of quantized llms. In Oh, A., Naumann, T., Globerson, A., Saenko, K., Hardt, M., and Levine, S. (eds.), *Advances in Neural Information Processing Systems*, volume 36, pp. 10088–10115. Curran Associates, Inc., 2023.

Frankle, J. and Carbin, M. The lottery ticket hypothesis: Finding sparse, trainable neural networks. In *International Conference on Learning Representations*, 2019. doi: 10.48550/arXiv.1803.03635.

Gim, I. and Ko, J. Memory-efficient dnn training on mobile devices. In *Proceedings of the 20th Annual International Conference on Mobile Systems, Applications and Services*, pp. 464–476, 2022.

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

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

Kim, S., Hooper, C., Gholami, A., Dong, Z., Li, X., Shen, S., Mahoney, M. W., and Keutzer, K. Squeezellm: Dense-and-sparse quantization. *arXiv preprint arXiv:2306.07629*, 2023.

Kingma, D. P. and Ba, J. Adam: A method for stochastic optimization. In *International Conference on Learning Representations*, 2015. doi: 10.48550/arXiv.1412.6980.

Li, L., Qian, S., Lu, J., Yuan, L., Wang, R., and Xie, Q. Transformer-lite: High-efficiency deployment of large language models on mobile phone gpus. *arXiv preprint arXiv:2403.20041*, 2024.

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

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*, 2019.

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

Liu, Z., Wang, J., Dao, T., Zhou, T., Yuan, B., Song, Z., Shrivastava, A., Zhang, C., Tian, Y., Re, C., et al. Deja vu: Contextual sparsity for efficient llms at inference time. In *International Conference on Machine Learning*, pp. 22137–22176. PMLR, 2023.

Liu, Z., Wang, G., Zhong, S. H., Xu, Z., Zha, D., Tang, R. R., Jiang, Z. S., Zhou, K., Chaudhary, V., Xu, S., et al. Winner-take-all column row sampling for memory efficient adaptation of language model. *Advances in Neural Information Processing Systems*, 36, 2024b.

Mairittha, N., Mairittha, T., and Inoue, S. Improving activity data collection with on-device personalization using fine-tuning. In *Adjunct Proceedings of the 2020 ACM International Joint Conference on Pervasive and Ubiquitous Computing and Proceedings of the 2020 ACM International Symposium on Wearable Computers*, pp. 255–260, 2020.

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

Malladi, S., Wettig, A., Yu, D., Chen, D., and Arora, S. A kernel-based view of language model fine-tuning. In *International Conference on Machine Learning*, pp. 23610–23641. PMLR, 2023b.

Merity, S., Xiong, C., Bradbury, J., and Socher, R. Pointer sentinel mixture models. In *International Conference on Learning Representations*, 2016.

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.

Panigrahi, A., Saunshi, N., Zhao, H., and Arora, S. Task-specific skill localization in fine-tuned language models. In *International Conference on Machine Learning*, pp. 27011–27033. PMLR, 2023.

Peng, Z., Yan, M., and Yin, W. Parallel and distributed sparse optimization. In *2013 Asilomar conference on signals, systems and computers*, pp. 659–646. IEEE, 2013.

Radford, A., Wu, J., Child, R., Luan, D., Amodei, D., Sutskever, I., et al. Language models are unsupervised multitask learners. *OpenAI blog*, 1(8):9, 2019.

Spall, J. C. Multivariate stochastic approximation using a simultaneous perturbation gradient approximation. *IEEE transactions on automatic control*, 37(3):332–341, 1992.

Tan, Z., Chen, T., Zhang, Z., and Liu, H. Sparsity-guided holistic explanation for llms with interpretable inference-time intervention. In *Proceedings of the AAAI Conference on Artificial Intelligence*, pp. 21619–21627, 2024a.

Tan, Z., Zeng, Q., Tian, Y., Liu, Z., Yin, B., and Jiang, M. Democratizing large language models via personalized parameter-efficient fine-tuning. *arXiv preprint arXiv:2402.04401*, 2024b.

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.

Xi, H., Li, C., Chen, J., and Zhu, J. Training transformers with 4-bit integers. *Advances in Neural Information Processing Systems*, 36:49146–49168, 2023.

Xia, H., Zheng, Z., Li, Y., Zhuang, D., Zhou, Z., Qiu, X., Li, Y., Lin, W., and Song, S. L. Flash-llm: Enabling cost-effective and highly-efficient large generative model inference with unstructured sparsity. In *Proceedings of the VLDB Endowment, Vol. 17, No. 2*, 2023. doi: 10.14778/3626292.3626303.

Xu, M., Qian, F., Mei, Q., Huang, K., and Liu, X. Deeptype: On-device deep learning for input personalization service with minimal privacy concern. *Proceedings of the ACM on Interactive, Mobile, Wearable and Ubiquitous Technologies*, 2(4):1–26, 2018.

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*, 2022.

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

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

Zhong, S., Zhang, G., Huang, N., and Xu, S. Revisit kernel pruning with lottery regulated grouped convolutions. In *International Conference on Learning Representations*, 2021.

Zhong, S. H., You, Z., Zhang, J., Zhao, S., LeClaire, Z., Liu, Z., Zha, D., Chaudhary, V., Xu, S., and Hu, X. One less reason for filter pruning: Gaining free adversarial robustness with structured grouped kernel pruning. *Advances in Neural Information Processing Systems*, 36, 2024.

Appendix

In Section A we describe all notations used in this paper. In Section B, we provide a high-level theoretical understanding and a convergence rate of optimizing our sensitive parameters via the ZO-SGD optimizer. In Section C, we discuss how our approach is positioned among sparsity-driven techniques in LLM community. In Section D, we include some additional experiments, describe all details in our experiments, and provide a high-level recommendation on how to efficiently implement our sensitive sparse ZO fine-tuning in forward passes of linear layers with existing quantization methods or training / inference workflow.

A. Notations

We present the notations used in this work as follows.

Table 2: Notations

Term/Symbol	**Explanation**
f	loss function
t	optimization timestep t
d	number of model parameters
d_{layer}	number of parameters in one linear layer. This means the total number of parameters per each linear layer as the number of rows times the number of columns in each linear layer.
$(\mathbf{x}_t, y_t)$	a data example sampled at timestep t as a pair of input vector and training target
$\mathbf{w}_t \in \mathbb{R}^d$	weight/parameter vector at optimization timestep t
$f(\mathbf{w}; (\mathbf{x}, y))$	training loss of $\mathbf{w}$ evaluated at a single data example $(\mathbf{x}, y)$
$\mathcal{F}(\mathbf{w}) = \mathbb{E}_{(\mathbf{x},y)} f(\mathbf{w}; (\mathbf{x}, y))$	full-batched training loss of $\mathbf{w}$
ϵ	a small perturbation scaling constant (close to 0)
$\mathbf{z}_t \in \mathbb{R}^d$	random Gaussian perturbation vector sampled at timestep t
$\hat{g}(\mathbf{w}, (\mathbf{x}, y), \mathbf{z})$	estimated ZO surrogate gradient for $\mathbf{w}$ with a data example $(\mathbf{x}, y)$ and a sampled Gaussian perturbation vector $\mathbf{z}$ (Definition 2.1)
η_t	learning rate for ZO-SGD optimizer (Definition 2.2) at timestep t
$\mathbf{m}_k \in \{0, 1\}^d$	a sensitive sparse mask with k nonzero entries (Definition 3.1)
$\mathbf{m}_{k,t} \in \{0, 1\}^d$	a sensitive sparse mask with k nonzero entries, and it is derived at optimization timestep t
$\mathbf{I}_d$	Identity matrix with shape $\mathbb{R}^{d\times d}$
$\tilde{\mathbf{I}}_{d,\mathbf{m}_k}$	$\tilde{\mathbf{I}}_{d,\mathbf{m}_k}$ is equal to the identity matrix $\mathbf{I}_d$ with the main diagonal masked by $\mathbf{m}_k$
$\bar{\mathbf{z}}_t = \mathbf{z}_t \odot \mathbf{m}_k$	a sampled Gaussian perturbation vector $\mathbf{z}_t$ at timestep t that is masked by $\mathbf{m}_k$ Notice that $\bar{\mathbf{z}}$ is equivalent as being sampled from $\mathcal{N}(\mathbf{0}_d, \tilde{\mathbf{I}}_{d,\mathbf{m}_k})$
$\mathbf{1}_d$	a vector of size d with all entries equal to 1
Tr	trace operation
$Q(\mathbf{w})$	parameter vector $\mathbf{w}$ that is quantized by Q
$\mathbf{F}$	(true) Fisher information matrix
$\hat{\mathbf{F}}$	empirical Fisher information matrix
p_{LLM}	LLM as a probabilistic model
$p_{\mathcal{D}}$	true data distribution
$\mathbf{w}_{\text{sparse}} = \mathbf{w} \odot \mathbf{m}_k$	sensitive parameters with positions as the nonzero entries sensitive sparse mask $\mathbf{m}_k$ (Equation 5)
L	Lipschitz constant in Assumption B.3
μ	PL condition number in Assumption B.4
σ^2	stochastic gradient error term in Assumption B.2
W_K	weight matrix of linear projection for the key embedding matrix K in attention layers
W_V	weight matrix of linear projection for the value embedding matrix V in attention layers

B. Theory

In Section B.1, we provide a high-level theoretical understanding of our sensitive parameters. In Section B.2, we include the assumption and exact proof on the convergence rate (Theorem B.1)

B.1. Theoretical intuition

The theoretical support of sensitive parameters can be derived from the lens of SPSA gradient estimator and Fisher information matrix as follows:

- **Maximum zeroth-order loss value changes, from the lens of SPSA estimator.**
The square (account for negativity) of loss value difference for $\hat{g}_{\mathbf{w}}(\mathbf{w}_t, (\mathbf{x}_t, y_t), \bar{\mathbf{z}}_t)$ is as follows:

$$\mathbb{E}_{\bar{\mathbf{z}}}\{f(\mathbf{w}+\epsilon\bar{\mathbf{z}};(\mathbf{x},y)) - f(\mathbf{w}-\epsilon\bar{\mathbf{z}};(\mathbf{x},y))\}^2 \approx \mathbb{E}_{\bar{\mathbf{z}}}\{2\epsilon\bar{\mathbf{z}}^\top \nabla_{\mathbf{w}} f(\mathbf{w};(\mathbf{x},y))\}^2 \tag{7}$$
$$= 4\epsilon^2 \|\mathbf{m}_k \odot \nabla_{\mathbf{w}} f(\mathbf{w};(\mathbf{x},y))\|^2 \tag{8}$$

Since by Definition 3.1 our sensitive mask would maximize $\|\mathbf{m}_k \odot \nabla_{\mathbf{w}} f(\mathbf{w};(\mathbf{x},y))\|^2$ for a given sparsity ratio, we would expect our sensitive mask to *maximize* the magnitude of the loss value difference *for any given sparsity ratio*.
- **Maximum coverage of Hessian diagonal, from the lens of Fisher matrix.**
LLMs are often pre-trained on large text corpus[2] to reach low perplexity before entering the fine-tuning stage. In this case, we would assume $p_{\text{LLM}}(y|\mathbf{x}) \sim p_{\mathcal{D}}(y|\mathbf{x})$, which implies the empirical Fisher $\hat{\mathbf{F}}$ should be close to the (true) Fisher matrix $\mathbf{F}$ as follows:

$$\mathbf{F} = \mathbb{E}_{\mathbf{x}\sim p_{\mathcal{D}}, \hat{y}\sim p_{\text{LLM}}(\cdot|\mathbf{x})} \nabla_{\mathbf{w}} \log p_{\text{LLM}}(\hat{y}|\mathbf{x})(\nabla_{\mathbf{w}} \log p_{\text{LLM}}(\hat{y}|\mathbf{x}))^\top \tag{9}$$
$$\approx \hat{\mathbf{F}} = \mathbb{E}_{(\mathbf{x},y)\sim p_{\mathcal{D}}} \nabla_{\mathbf{w}} \log p_{\text{LLM}}(y|\mathbf{x})(\nabla_{\mathbf{w}} \log p_{\text{LLM}}(y|\mathbf{x}))^\top \tag{10}$$

As we assume the empirical Fisher matrix approximates Fisher, which also approximates the Hessian, and empirical Fisher's diagonal is *equal* to the coordinate-wise gradient square vector when computing with downstream task-specific loss, our sensitive parameters would cover a large fraction of the largest Hessian diagonal entries.

B.2. Theoretical Convergence Rate

We would investigate the theoretical convergence of sensitive sparse ZO-SGD on sensitive parameters under the non-convex optimization settings. Our assumptions are included in Appendix B.4.

Theorem B.1 (**Convergence rate of sensitive sparse ZO-SGD (Definition 3.2)**). *If we pick $\eta_t = 1/(L(k+2))$, under Assumptions B.2 (bounded gradient error), B.3 (Lipschitz smoothness), and B.5 (sparse sensitive parameters), we would have*

$$\frac{1}{T}\sum_{t=0}^{T-1} \mathbb{E}_{\bar{\mathbf{z}},(\mathbf{x},y)} \|\nabla_{\mathbf{w}} \mathcal{F}(\mathbf{w}_t)\|^2 \leq O\left(\frac{k}{c} \cdot \frac{L}{T}\right)(\mathcal{F}(\mathbf{w}_0) - \mathcal{F}^*) + 3\sigma^2. \tag{11}$$

Moreover, if we still pick $\eta_t = 1/(L(k+2))$, with an extra Assumption B.4 (P.L. condition), we would have

$$\mathbb{E}_{\bar{\mathbf{z}},(\mathbf{x},y)}\{\mathcal{F}(\mathbf{w}_T) - \mathcal{F}^*\} \leq \left(1 - O\left(\frac{\mu}{L} \cdot \frac{c}{k}\right)\right)^T (\mathcal{F}(\mathbf{w}_0) - \mathcal{F}^*) + \frac{3\sigma^2 c}{2L(k+2)}. \tag{12}$$

The proof for Inequality 11 is in Appendix B.4 and the proof for Inequality 12 is in Appendix B.5. If we choose $k = d$ and $c = 1$, both convergence rates trivially reduce to the standard zeroth-order convergence rate as $O(d/T) + O(\text{constant})$ and $O((1/d)^T) + O(\text{constant})$. As we assume $c \gg k/d$, we know $d \gg k/c$ and therefore both $O((k/c)(1/T))$ and $O((c/k)^T)$ are much lower than $O(d/T) + O(\text{constant})$ and $O((1/d)^T) + O(\text{constant})$ that zeroth-order method will yield.

We want to emphasize that our contributions are more on empirical LLM fine-tuning instead of general machine learning tasks, and in Section 4.2 we extensively compare our sparse ZO methods with other sparse ZO methods and we demonstrate its superiority during LLM fine-tuning. We do not use the strict "local r-effective rank" assumption that Malladi et al. (2023a) uses, and our Assumption B.5 can be easily observed empirically in Figure 1. Liu et al. (2024a) and Ohta et al. (2020) also provide similar analysis on the convergence. However, they do not include our sensitive sparse mask in their studies.

[2]Here we assume data examples $(\mathbf{x}, y) \sim p_{\mathcal{D}}$ in fine-tuning datasets after verbalization would also appear in the large text corpus during pre-training.

B.3. Assumptions

We start with listing standard assumptions in nonconvex optimization literature:

Assumption B.2 (Bounded stochastic gradient errors). For any data example $(\mathbf{x}, y) \in \mathcal{D}$ and for any $\mathbf{w} \in \mathbb{R}^d$, denote the full-batched loss function $\mathcal{F}(\mathbf{w}) = \mathbb{E}_{(\mathbf{x},y)\in\mathcal{D}} f(\mathbf{w}; (\mathbf{x}, y))$, we have

$$\|\nabla_{\mathbf{w}} f(\mathbf{w}; (\mathbf{x}, y)) - \nabla_{\mathbf{w}} \mathcal{F}(\mathbf{w})\|^2 \leq \sigma^2. \tag{13}$$

Assumption B.3 (Lipschitz smoothness). We assume that $f(\mathbf{w}, \mathbf{x})$ is L-Lipschitz smooth ($L > 0$): for any $\mathbf{w}, \mathbf{w}' \in \mathbb{R}^d$,

$$\|\nabla_{\mathbf{w}} f(\mathbf{w}; (\mathbf{x}, y)) - \nabla_{\mathbf{w}} f(\mathbf{w}'; (\mathbf{x}, y))\| \leq L\|\mathbf{w} - \mathbf{w}'\|. \tag{14}$$

Assumption B.4 (PL inequality). We assume that $\mathcal{F}(\mathbf{w})$ fulfills the Polyak-Lojasiewicz (PL) condition: there exists some $\mu > 0$, for any $\mathbf{w} \in \mathbb{R}^d$

$$\frac{1}{2}\|\nabla_{\mathbf{w}} \mathcal{F}(\mathbf{w})\|^2 \geq \mu(\mathcal{F}(\mathbf{w}) - \mathcal{F}^*), \quad \mathcal{F}^* \text{ is the minimum value } \mathcal{F}^* = \inf_{\mathbf{w}} \mathcal{F}(\mathbf{w}). \tag{15}$$

Inspired by Figure 4, we would assume the sensitive parameters of $\mathbf{w}$ are sparse.

Assumption B.5 (Sensitive parameters are sparse). We assume at timestep t $\exists \mathbf{m}_t \in \{0, 1\}^d$ with the number of nonzero entries as k, $\exists c \in [0, 1]$ such that

$$\|\mathbf{m}_t \odot \nabla_{\mathbf{w}} f(\mathbf{w}_t; (\mathbf{x}_t, y_t))\|^2 = c\|\nabla_{\mathbf{w}} f(\mathbf{w}_t; (\mathbf{x}_t, y_t))\|^2.$$

Here we assume $c \gg k/d$. [3]

B.4. Proof for Equation 11, Theorem B.1

We will start with formulating the expectation of sensitive sparse ZO surrogate gradient norm square in terms of its corresponding stochastic gradient norm square.

Lemma B.6 (Sensitive sparse ZO surrogate gradient norm square).

$$\mathbb{E}_{\bar{\mathbf{z}}}[\|\hat{g}(\mathbf{w}_t, (\mathbf{x}_t, y_t), \bar{\mathbf{z}}_t)\|^2] = (2 + k)c\|\nabla_{\mathbf{w}} f(\mathbf{w}, (\mathbf{x}_t, y_t))\|^2$$

Proof for Lemma B.6. We know that our $\bar{\mathbf{z}}$ can be considered as being sampled from $\mathcal{N}(\mathbf{0}, \tilde{\mathbf{I}}_{d,\mathbf{m}_k})$ where $\tilde{\mathbf{I}}_{d,\mathbf{m}_k}$ is the identity matrix $\mathbf{I}_d$ with the main diagonal masked by $\mathbf{m}_k$.

We expand the sensitive sparse ZO surrogate gradient covariance matrix $\mathbb{E}_{\bar{\mathbf{z}}}\hat{g}(\mathbf{w}, (\mathbf{x}, y), \bar{\mathbf{z}})\hat{g}(\mathbf{w}, (\mathbf{x}, y), y), \bar{\mathbf{z}})^\top$ as follows:

$$\begin{aligned}
&\mathbb{E}_{\bar{\mathbf{z}}}\hat{g}(\mathbf{w}, (\mathbf{x}, y), \bar{\mathbf{z}})\hat{g}(\mathbf{w}, (\mathbf{x}, y), \bar{\mathbf{z}})^\top \\
&= \mathbb{E}_{\bar{\mathbf{z}}_i}[\bar{\mathbf{z}}_i\bar{\mathbf{z}}_i^\top \left((\mathbf{m}_k \odot \nabla_{\mathbf{w}} f(\mathbf{w}; (\mathbf{x}, y)))(\mathbf{m}_k \odot \nabla_{\mathbf{w}} f(\mathbf{w}; (\mathbf{x}, y)))^\top\right) \bar{\mathbf{z}}_i\bar{\mathbf{z}}_i^\top] \\
&= 2\left((\mathbf{m}_k \odot \nabla_{\mathbf{w}} f(\mathbf{w}; (\mathbf{x}, y)))(\mathbf{m}_k \odot \nabla_{\mathbf{w}} f(\mathbf{w}; (\mathbf{x}, y)))^\top\right) + \|\mathbf{m}_k \odot \nabla_{\mathbf{w}} f(\mathbf{w}; (\mathbf{x}, y))\|^2 \tilde{\mathbf{I}}_{d,\mathbf{m}_k}
\end{aligned}$$

Then the sensitive sparse ZO surrogate gradient norm square is the square of the *diagonal* of its corresponding covariance matrix:

$$\begin{aligned}
\mathbb{E}_{\bar{\mathbf{z}}}[\|\hat{g}(\mathbf{w}_t, \mathbf{x}_t, \bar{\mathbf{z}}_t)\|^2] &= \text{diag}\left(\mathbb{E}_{\bar{\mathbf{z}}}\hat{g}(\mathbf{w}, (\mathbf{x}, y), \bar{\mathbf{z}})\hat{g}(\mathbf{w}, (\mathbf{x}, y), y), \bar{\mathbf{z}})^\top\right)^2 \\
&= 2c\|\nabla_{\mathbf{w}} f(\mathbf{w}, (\mathbf{x}_t, y_t))\|^2 + kc\|\nabla_{\mathbf{w}} f(\mathbf{w}, (\mathbf{x}_t, y_t))\|^2 \\
&= (2 + k)c\|\nabla_{\mathbf{w}} f(\mathbf{w}, (\mathbf{x}_t, y_t))\|^2
\end{aligned}$$

□

Then we are in good shape of deriving the convergence rate under the Lipschitz smoothness condition:

[3] From Figure 4, we know that for $c \sim 0.5$, we only need $k/d \sim 0.001$. In this case $k/c \sim 0.002d$.

Proof for Equation 11, Theorem B.1.

$$\begin{aligned}
f(\mathbf{w}_{t+1}, \mathbf{x}_t) &\leq f(\mathbf{w}_t; (\mathbf{x}_t, y_t)) + \langle \nabla f(\mathbf{w}_t; (\mathbf{x}_t, y_t)), \mathbf{w}_{t+1} - \mathbf{w}_t \rangle + \frac{L}{2}\|\mathbf{w}_{t+1} - \mathbf{w}_t\|^2 \\
&\leq f(\mathbf{w}_t; (\mathbf{x}_t, y_t)) - \eta_t \langle \nabla f(\mathbf{w}_t; (\mathbf{x}_t, y_t)), \hat{g}(\mathbf{w}_t, \mathbf{x}_t, \bar{\mathbf{z}}_t) \rangle + \frac{L\eta_t^2}{2}\|\hat{g}(\mathbf{w}_t, \mathbf{x}_t, \bar{\mathbf{z}}_t)\|^2 \\
\mathbb{E}_{\bar{\mathbf{z}}} f(\mathbf{w}_{t+1}, \mathbf{x}_t) &\leq \mathbb{E}_{\bar{\mathbf{z}}} f(\mathbf{w}_t; (\mathbf{x}_t, y_t)) - \eta_t \mathbb{E}_{\bar{\mathbf{z}}}\|\mathbf{m}_{k,t} \odot \nabla f(\mathbf{w}_t; (\mathbf{x}_t, y_t))\|^2 + \frac{L\eta_t^2}{2}\mathbb{E}_{\bar{\mathbf{z}}}\|\hat{g}(\mathbf{w}_t, \mathbf{x}_t, \bar{\mathbf{z}})\|^2 \\
\mathbb{E}_{\bar{\mathbf{z}}} f(\mathbf{w}_{t+1}, \mathbf{x}_t) &\leq \mathbb{E}_{\bar{\mathbf{z}}} f(\mathbf{w}_t; (\mathbf{x}_t, y_t)) - c\eta_t \mathbb{E}_{\bar{\mathbf{z}}}\|\nabla f(\mathbf{w}_t; (\mathbf{x}_t, y_t))\|^2 + \frac{L\eta_t^2}{2}c(k+2)\mathbb{E}_{\bar{\mathbf{z}}}\|\nabla_{\mathbf{w}} f(\mathbf{w}_t; (\mathbf{x}_t, y_t))\|^2 \\
\mathbb{E}_{\bar{\mathbf{z}},(\mathbf{x},y)} \mathcal{F}(\mathbf{w}_{t+1}) &\leq \mathbb{E}_{\bar{\mathbf{z}},(\mathbf{x},y)}\{\mathcal{F}(\mathbf{w}_t) - c\eta_t\|\nabla_{\mathbf{w}}\mathcal{F}(\mathbf{w}_t)\|^2 + c\sigma^2\eta_t + \frac{L\eta_t^2}{2}c(k+2)\|\nabla_{\mathbf{w}}\mathcal{F}(\mathbf{w}_t)\|^2 + \frac{L\eta_t^2}{2}c(k+2)\sigma^2\} \\
\mathbb{E}_{\bar{\mathbf{z}},(\mathbf{x},y)} \mathcal{F}(\mathbf{w}_{t+1}) &\leq \mathbb{E}_{\bar{\mathbf{z}},(\mathbf{x},y)}\{\mathcal{F}(\mathbf{w}_t) - \left(c\eta_t - \frac{L\eta_t^2}{2}c(k+2)\right)\|\nabla_{\mathbf{w}}\mathcal{F}(\mathbf{w}_t)\|^2 + \left(c\sigma^2\eta_t + \frac{L\eta_t^2}{2}c(k+2)\sigma^2\right)\}
\end{aligned}$$

Denote $\alpha = Lc(k+2)$, we will have

$$\mathbb{E}_{\bar{\mathbf{z}},(\mathbf{x},y)} \mathcal{F}(\mathbf{w}_{t+1}) \leq \mathbb{E}_{\bar{\mathbf{z}},(\mathbf{x},y)}\{\mathcal{F}(\mathbf{w}_t) - \eta_t\left(c - \frac{\alpha}{2}\eta_t\right)\|\nabla_{\mathbf{w}}\mathcal{F}(\mathbf{w}_t)\|^2 + \left(c\sigma^2\eta_t + \frac{\alpha}{2}\sigma^2\eta_t^2\right)\}$$

Set $\eta_t < \frac{c}{\alpha} = \frac{1}{L(k+2)}$, we have

$$\mathbb{E}_{\bar{\mathbf{z}},(\mathbf{x},y)} \mathcal{F}(\mathbf{w}_{t+1}) \leq \mathbb{E}_{\bar{\mathbf{z}},(\mathbf{x},y)}\{\mathcal{F}(\mathbf{w}_t) - \frac{c\eta_t}{2}\|\nabla\mathcal{F}(\mathbf{w}_t)\|^2 + \left(c\sigma^2\eta_t + \frac{\alpha}{2}\sigma^2\eta_t^2\right)\}$$

If we apply our sparse ZO update rule recursively for T steps,

$$\begin{aligned}
\frac{1}{T}\sum_{t=0}^{T-1}\mathbb{E}_{\bar{\mathbf{z}},(\mathbf{x},y)}\|\nabla_{\mathbf{w}}\mathcal{F}(\mathbf{w}_t)\|^2 &\leq \frac{2\alpha}{Tc^2}(\mathcal{F}(\mathbf{w}_0) - \mathcal{F}^*) + \frac{1}{T}\sum_{t=0}^{T-1}\frac{\left(c\sigma^2\eta_t + \frac{\alpha}{2}\sigma^2\eta_t^2\right)}{\frac{c\eta_t}{2}} \\
&\leq \frac{2\alpha}{Tc^2}(\mathcal{F}(\mathbf{w}_0) - \mathcal{F}^*) + (2\sigma^2 + \sigma^2) \\
&\leq \frac{2L(k+2)}{c}\frac{1}{T}(\mathcal{F}(\mathbf{w}_0) - \mathcal{F}^*) + 3\sigma^2 \\
&\leq O\left(\frac{k}{c}\cdot\frac{L}{T}\right)(\mathcal{F}(\mathbf{w}_0) - \mathcal{F}^*) + 3\sigma^2
\end{aligned}$$

□

B.5. Proof for Equation 12, Theorem B.1

We can derive a convergence rate of sensitive sparse ZO-SGD optimization method under PL inequality and Lipschitz-smoothness as follows (this proof resumes from our prior proof with the Lipschitz-smoothness condition alone):

Proof for Equation 12, Theorem B.1. Denote κ as the condition number $\kappa = \frac{\mu}{L}$.

$$\begin{aligned}
\mathbb{E}_{\bar{\mathbf{z}},(\mathbf{x},y)}\mathcal{F}(\mathbf{w}_{t+1}) &\leq \mathbb{E}_{\bar{\mathbf{z}},(\mathbf{x},y)}\{\mathcal{F}(\mathbf{w}_t) - \frac{c\eta_t}{2}\|\nabla\mathcal{F}(\mathbf{w}_t)\|^2 + \left(c\sigma^2\eta_t + \frac{\alpha}{2}\sigma^2\eta_t^2\right)\} \\
&\leq \mathbb{E}_{\bar{\mathbf{z}},(\mathbf{x},y)}\{\mathcal{F}(\mathbf{w}_t) - c\mu\eta_t(\mathcal{F}(\mathbf{w}_t) - \mathcal{F}^*) + \left(c\sigma^2\eta_t + \frac{\alpha}{2}\sigma^2\eta_t^2\right)\} \\
\mathbb{E}_{\bar{\mathbf{z}},(\mathbf{x},y)}\{\mathcal{F}(\mathbf{w}_{t+1}) - \mathcal{F}^*\} &\leq \mathbb{E}_{\bar{\mathbf{z}},(\mathbf{x},y)}\{(\mathcal{F}(\mathbf{w}_t) - \mathcal{F}^*) - c\mu\eta_t(\mathcal{F}(\mathbf{w}_t) - \mathcal{F}^*) + \left(c\sigma^2\eta_t + \frac{\alpha}{2}\sigma^2\eta_t^2\right)\} \\
\mathbb{E}_{\bar{\mathbf{z}},(\mathbf{x},y)}\{\mathcal{F}(\mathbf{w}_{t+1}) - \mathcal{F}^*\} &\leq \mathbb{E}_{\bar{\mathbf{z}},(\mathbf{x},y)}\{(\mathcal{F}(\mathbf{w}_t) - \mathcal{F}^*) - c\mu\eta_t(\mathcal{F}(\mathbf{w}_t) - \mathcal{F}^*) + \left(c\sigma^2\eta_t + \frac{\alpha}{2}\sigma^2\eta_t^2\right)\}
\end{aligned}$$

Plugging in $\eta_t \leq \dfrac{c}{\alpha}$ and applying recursively for T iterations.

$$\begin{aligned}
\mathbb{E}_{\bar{\mathbf{z}},(\mathbf{x},y)}\{\mathcal{F}(\mathbf{w}_T) - \mathcal{F}^*\} &\leq (1 - \frac{c\kappa}{(k+2)})^T(\mathcal{F}(\mathbf{w}_0) - \mathcal{F}^*) + \frac{3\sigma^2c^2}{2\alpha} \\
&\leq (1 - \frac{c\kappa}{(k+2)})^T(\mathcal{F}(\mathbf{w}_0) - \mathcal{F}^*) + \frac{3\sigma^2c}{2L(k+2)} \\
&\leq \left(1 - O\left(\frac{\mu}{L}\cdot\frac{c}{k}\right)\right)^T(\mathcal{F}(\mathbf{w}_0) - \mathcal{F}^*) + \frac{3\sigma^2c}{2L(k+2)}
\end{aligned}$$

□

C. Sparsity in LLMs

Sparsity-driven techniques are widely adopted in improving ML model's efficiency (Tan et al., 2024a; Xia et al., 2023; Liu et al., 2023; Peng et al., 2013; Frankle & Carbin, 2019) and robustness (Zhong et al., 2024; 2021). Frankle & Carbin (2019) showed that within large feed-forward networks, there exists a subnetwork that, when trained in isolation, can achieve test accuracy comparable to that of the original network. In the foundation models era, Liu et al. (2023) demonstrated that transformer-based models, such as OPT (Zhang et al., 2022), exhibit great sparsity ($\geq 95\%$) in activations. Moreover, Panigrahi et al. (2023) discovered that for RoBERTa (Liu et al., 2019), fine-tuning a very small subset of parameters ($\sim 0.01\%$) can yield performance exceeding 95% of that achieved by full fine-tuning.

In the context of ZO optimization, Liu et al. (2024a) and Zhang et al. (2024) also suggest that sparsity would potentially accelerate ZO optimization convergence. We believe that *ZO has an intrinsic need for sparse training*, as the procedure of ZO gradient estimator usually requires *nearly uniform coordinate-wise scale (in expectation)* perturbation which grows with d. In tradition, people usually resolve this with knowledge from parameter-wise loss curvature heterogeneity (replace $\mathbf{z}$ with $\Sigma^{1/2}\mathbf{z}$ where $\Sigma^{1/2}$ serves as a Hessian-informed preconditioner) (Ye et al., 2018; Zhao et al., 2024). However, they do not provide a comprehensive investigation on massive parameter models like LLMs. In particular, we also observe that during first-order (FO) fine-tuning of LLMs, *the FO gradient can be quite sparse*. We will elaborate more on this insight in the following section (see Figure 1 and Figure 4). We would like to explore how sparsity can benefit the ZO LLM fine-tuning.

D. Supplementary Experiment Details

D.1. On-device memory constraints

We include a table of common memory constraints imposed by edge or mobile devices as Table 3. We can find that a wide range of these devices impose a memory constraint of **8 GiB** as our main main constraint that we consider when we develop our on-device personalization recipe in Section 3.2.

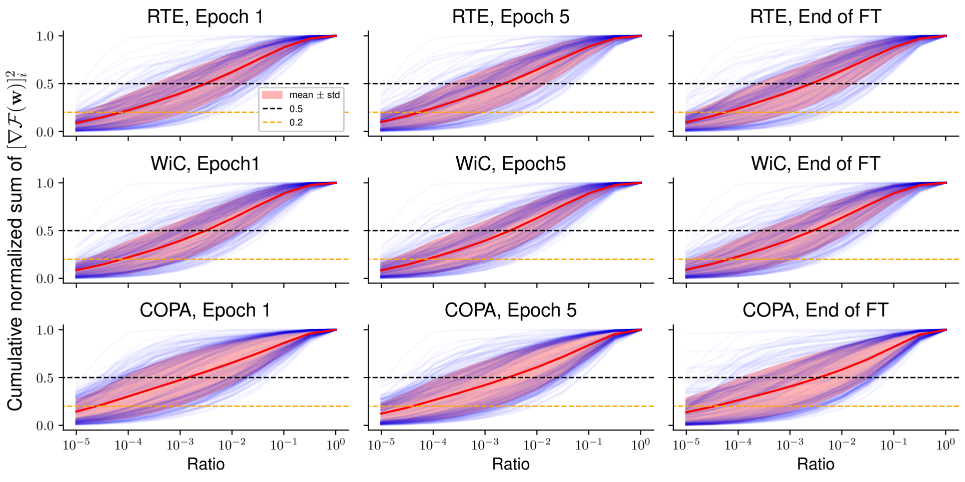
Table 3: Device memory of some mobile devices or consumer-graded GPUs.

Devices	Memory
Nvidia GeForce GTX 1080 Ti	11 GiB
Nvidia GeForce RTX 3060 Ti	8 GiB
Nvidia Jetson TX2	8 GiB
OPPO Find X7 Ultra (Li et al., 2024)	12 GiB
Samsung Galaxy S10 with Mali-G76 GPU (Gim & Ko, 2022)	8 GiB

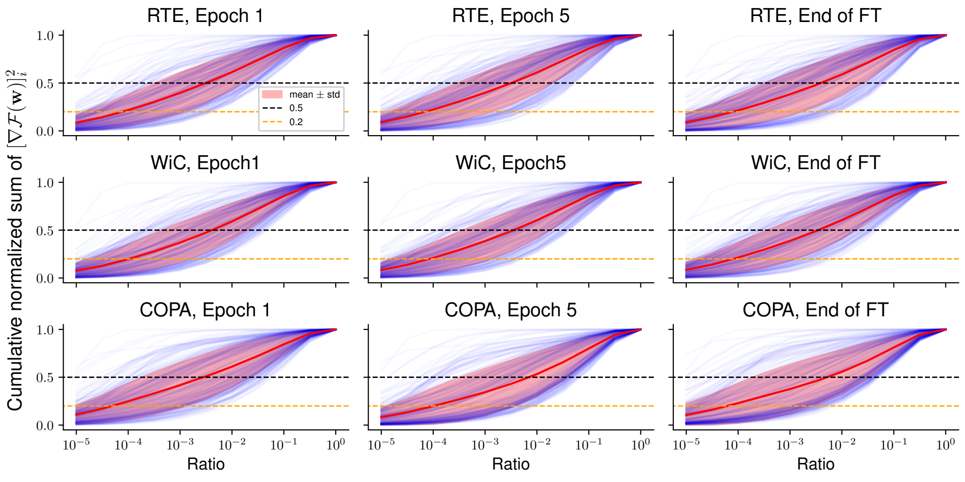
D.2. Gradient sparsity during LLM fine-tuning

In Figure 1, we explore the FO gradient sparsity of Llama2-7B during fine-tuning (at Epoch 5). Here we follow the identical setting and plot the FO gradient sparsity for Llama2-7B, Mistral-7B, and OPT-6.7B during epoch 1, 5, and 10 (end of fine-tuning).

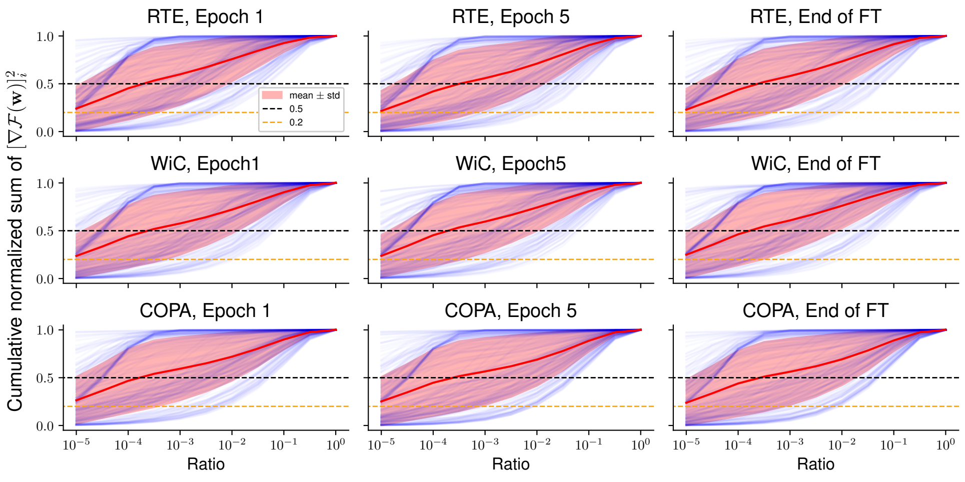
We observe that the gradient sparsity is exhibited throughout the fine-tuning with slightly increasing towards the end. OPT-6.7B which uses ReLU as the activation function would demonstrate greater sparsity across tasks compared with Llama2-7B and Mistral-7B which uses SwiGLU and SiLU respectively. Nevertheless, the gradient sparsity pattern holds across architectures, tasks, and fine-tuning time in general.



(a) Llama2-7B



(b) Mistral-7B

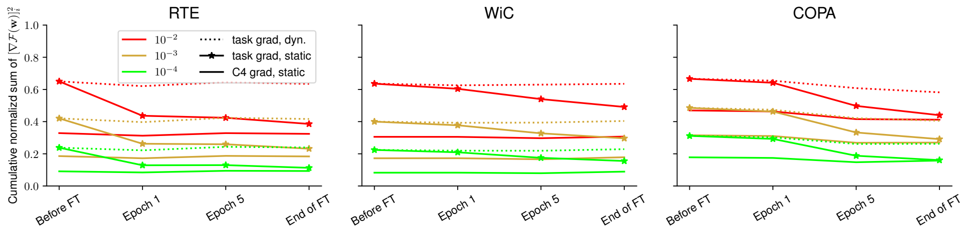


(c) OPT-6.7B

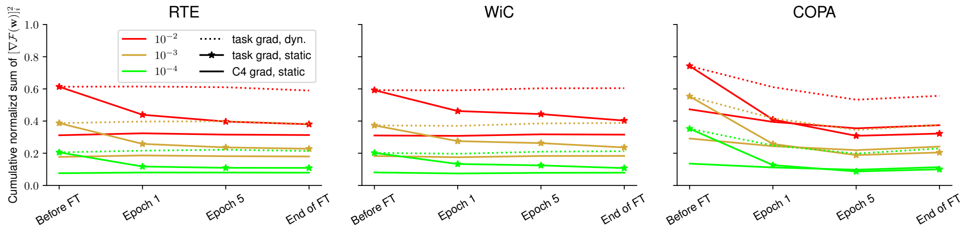
Figure 4: Cumulative normalized sum of coordinate-wise gradient square $[\nabla \mathcal{F}(\mathbf{w})]_i^2$ of linear layers for Llama2-7B (subfigure 4a), Mistral-7B (subfigure 4b), and OPT-6.7B (subfigure 4c) across RTE, WiC, and COPA tasks during FO-SGD fine-tuning. For each linear layer, we first sort parameters by the decreasing order of their gradient square value $[\nabla \mathcal{F}(\mathbf{w})]_i^2, i \in [d_{\text{layer}}]$, and we take the cumulative sum and normalize it to draw a blue curve, and the red-shaded region is the mean ± std of all blue curves.

D.3. Transferability of gradient features from pre-training datasets to downstream tasks

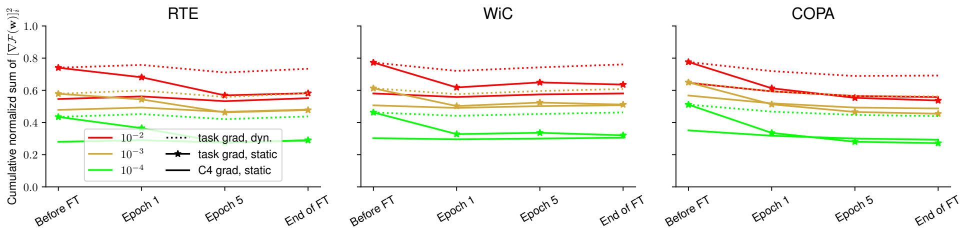
Here we will explore the transferability of gradient features from pre-training datasets (C4) to downstream tasks, as shown in Figure 5. As there are *no* solid lines (top-(1e-2,1e-3,1e-4)) parameters with C4 gradient entries prior to fine-tuning) vanish to 0, we know the transferability of gradient features from C4 datasets to downstream datasets hold across models and downstream tasks. In this case, sensitive parameters determined from C4 gradients would still be similar to sensitive parameters determined from downstream task-specific gradients across models.



(a) Llama2-7B



(b) Mistral-7B



(c) OPT-6.7B

Figure 5: Cumulative normalized gradient square values of Llama2-7B (subfigure 5a), Mistral-7B (subfigure 5b), and OPT-6.7B (subfigure 5c)'s linear layers during FO fine-tuning. For a given model and training checkpoint, we report the average value across all linear layers as a line in each subfigure. For each line, the colors represent the fraction of parameters (1e-2,1e-3,1e-4) and the line style represents the category. "task grad, dyn." refers to the sensitive parameters selected at the given timestep (x-axis), and "task grad, static" refers to the sensitive parameters selected before fine-tuning. "C4 grad, static" refers to the sensitive parameters selected with gradients taken from causal language modeling on C4 datasets, and we keep it unchanged during fine-tuning.

D.4. More experiments

We include the complete version of Table 1 as Table 4, a complete version of subfigure 3a and 3b in subfigure 7a and 7b. We also discuss wall-clock time convergence speedup and iteration-wise training / token-wise inference speedup by optimizing only 0.1% parameters by sensitive sparse ZO in Figure 6.

D.4.1. On-device personalization

Here we include the complete version of Table 1 as Table 4. The analysis of this Table 4 is the same as our discussion in Section 4.1.

Table 4: Performance of difference methods on Llama2-7B fine-tuning tasks. In the first column, "Q" means the full model is quantized with 4-bit quantization method (SqueezeLLM (Kim et al., 2023)), and "ZO" means the model is finetuned with ZO-SGD optimizer. For each cell, we use the same hyperparameters and repeat it with 3 random seeds. We report the average and standard deviation of test set accuracy in the format of $\mathbf{mean}_{\mathbf{std}}$. In last 2 columns, "Acc" means the average test set accuracy and "Rank" means the average rank among all methods across tasks.

(a) **Llama2-7B**

	Methods	SST-2	RTE	CB	BoolQ	WSC	WiC	COPA	**Acc**	**Rank**
Q, ZO	**Sensitive (C4, static)**	$\mathbf{94.7}_{0.4}$	$\mathbf{74.7}_{1.2}$	$\mathbf{66.7}_{2.2}$	$\mathbf{83.0}_{0.5}$	$57.4_{3.9}$	$\mathbf{65.2}_{0.9}$	$85.0_{2.2}$	**75.2**	**2.43**
	LoRA	$93.8_{0.6}$	$64.7_{1.1}$	$64.9_{4.7}$	$79.7_{1.1}$	$\mathbf{61.5}_{2.1}$	$59.8_{0.1}$	$\mathbf{85.7}_{0.5}$	72.9	4.29
	Prefix	$80.5_{4.3}$	$65.5_{1.2}$	$63.1_{3.0}$	$80.3_{0.2}$	$54.5_{11.4}$	$58.3_{1.3}$	$82.0_{0.8}$	69.2	5.86
ZO	**Sensitive (task, static)**	$\mathbf{94.8}_{0.1}$	$\mathbf{73.6}_{0.9}$	$\mathbf{69.1}_{2.2}$	$\mathbf{83.5}_{0.8}$	$57.4_{4.7}$	$64.2_{1.1}$	$\mathbf{83.7}_{2.4}$	**75.2**	**2.29**
	Random (static)	$94.1_{0.3}$	$68.0_{1.7}$	$64.9_{3.4}$	$77.0_{0.7}$	$\mathbf{59.6}_{3.6}$	$\mathbf{64.8}_{1.1}$	$83.3_{1.7}$	73.1	4.14
	Full fine-tuning	$94.6_{0.5}$	$73.3_{5.1}$	$66.7_{0.8}$	$81.9_{0.8}$	$58.0_{4.3}$	$61.9_{0.2}$	$82.7_{1.7}$	74.2	3.57
	Zero-shot	$89.0_{0.0}$	$57.8_{0.0}$	$32.1_{0.0}$	$69.9_{0.2}$	$50.2_{0.0}$	$36.5_{0.0}$	$79.0_{0.0}$	59.2	7.29
	ICL	$94.8_{0.2}$	$71.5_{4.3}$	$72.6_{15.2}$	$77.5_{4.6}$	$53.2_{1.1}$	$61.1_{4.3}$	$87.0_{2.2}$	74.0	3.43

(b) **Mistral-7B**

	Methods	SST-2	RTE	CB	BoolQ	WSC	WiC	COPA	**Acc**	**Rank**
Q, ZO	**Sensitive (C4, static)**	$\mathbf{94.0}_{0.3}$	$\mathbf{74.2}_{2.7}$	$\mathbf{70.2}_{2.2}$	$\mathbf{75.1}_{2.4}$	$59.6_{4.9}$	$\mathbf{61.2}_{0.9}$	$\mathbf{88.3}_{1.2}$	**74.7**	**2.86**
	LoRA	$\mathbf{94.0}_{0.4}$	$65.3_{1.3}$	$64.9_{4.5}$	$70.3_{3.7}$	$\mathbf{60.9}_{3.7}$	$61.1_{0.4}$	$\mathbf{88.3}_{0.5}$	72.1	3.57
	Prefix	$86.9_{2.1}$	$57.3_{1.4}$	$63.7_{5.9}$	$62.2_{0.9}$	$60.3_{4.6}$	$49.0_{0.3}$	$81.3_{1.7}$	65.8	4.86
ZO	**Sensitive (task, static)**	$\mathbf{94.7}_{0.3}$	$\mathbf{77.1}_{0.9}$	$\mathbf{69.0}_{0.8}$	$\mathbf{78.4}_{2.2}$	$\mathbf{58.0}_{4.3}$	$61.4_{0.2}$	$\mathbf{89.3}_{1.3}$	**75.4**	**1.86**
	Random (static)	$87.9_{1.9}$	$50.2_{0.8}$	$66.1_{4.4}$	$60.6_{1.7}$	$57.6_{1.4}$	$57.3_{0.8}$	$82.3_{1.7}$	66.0	5.29
	Full fine-tuning	$94.6_{0.1}$	$74.6_{2.1}$	$68.8_{6.2}$	$76.6_{0.2}$	$54.8_{6.2}$	$\mathbf{62.6}_{0.5}$	$88.3_{0.5}$	74.3	2.86
	Zero-shot	$54.8_{0.0}$	$50.5_{0.0}$	$37.5_{0.0}$	$43.4_{1.8}$	$50.8_{0.0}$	$39.4_{0.0}$	$78.0_{0.0}$	50.6	7.00
	ICL	$60.7_{16.7}$	$55.2_{4.7}$	$33.3_{13.1}$	$46.8_{6.5}$	$50.4_{0.6}$	$63.8_{0.9}$	$88.7_{0.5}$	57.0	5.43

(c) **OPT-6.7B**

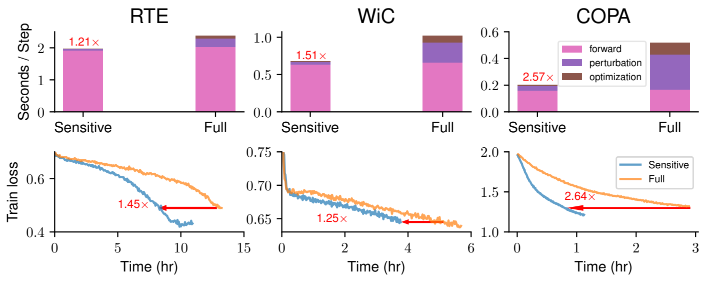
	Methods	SST-2	RTE	CB	BoolQ	WSC	WiC	COPA	**Acc**	**Rank**
Q, ZO	**Sensitive (C4, static)**	$\mathbf{94.9}_{0.5}$	$\mathbf{72.8}_{3.6}$	$\mathbf{83.3}_{5.1}$	$\mathbf{73.1}_{0.9}$	$\mathbf{59.3}_{5.3}$	$\mathbf{60.9}_{0.4}$	$\mathbf{84.0}_{1.4}$	**75.5**	**1.29**
	LoRA	$94.2_{0.2}$	$69.6_{1.6}$	$69.0_{1.7}$	$69.6_{2.0}$	$57.1_{9.1}$	$57.2_{0.8}$	$83.0_{2.2}$	71.4	4.57
	Prefix	$93.3_{0.4}$	$71.2_{1.0}$	$72.0_{1.7}$	$68.9_{2.8}$	$62.5_{2.4}$	$59.4_{0.5}$	$80.0_{2.4}$	72.5	4.14
ZO	**Sensitive (task, static)**	$\mathbf{94.5}_{0.4}$	$\mathbf{75.5}_{1.4}$	$\mathbf{82.1}_{3.6}$	$\mathbf{72.5}_{0.8}$	$\mathbf{57.4}_{5.2}$	$\mathbf{60.6}_{1.4}$	$\mathbf{83.3}_{1.7}$	**75.1**	**2.14**
	Random (static)	$87.3_{2.0}$	$68.4_{1.7}$	$70.6_{6.3}$	$66.0_{1.0}$	$58.0_{7.0}$	$56.4_{1.3}$	$79.0_{0.8}$	69.4	5.71
	Full fine-tuning	$94.4_{0.3}$	$72.7_{1.2}$	$79.8_{3.0}$	$72.1_{1.2}$	$\mathbf{57.4}_{4.6}$	$60.2_{0.9}$	$82.3_{2.6}$	74.1	3.29
	Zero-shot	$61.0_{0.0}$	$60.7_{0.0}$	$46.4_{0.0}$	$55.7_{1.0}$	$55.5_{0.0}$	$36.5_{0.0}$	$77.0_{0.0}$	56.1	7.71
	ICL	$74.0_{14.6}$	$65.8_{11.2}$	$54.8_{5.9}$	$67.9_{2.1}$	$53.2_{1.7}$	$41.0_{4.5}$	$80.7_{2.9}$	62.5	6.57

D.4.2. Wall-Clock Time Efficiency

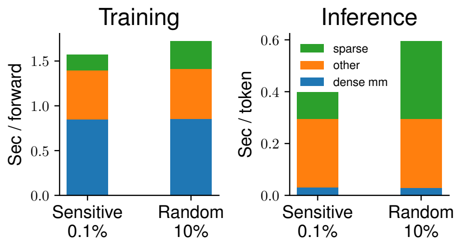
By employing parameter-efficient ZO fine-tuning with extreme sparsity, we also achieve 1.2 - 2.5$\times$ wall-clock time convergence speedup compared with ZO full fine-tuning as we nearly eliminate the ZO perturbation and optimizer update time, as Figure 6a shows. This also boosts the GPU utilization rate as large-batched ZO forward is often compute-bounded while the perturbation and optimization steps are often memory-bounded. Furthermore, the reduced memory footprint of parameter-efficient ZO fine-tuning allows for training larger models on the same hardware, potentially leading to even better performance. As a result, we answer this question that optimizing extremely sparse and fixed parameters leads to substantial iteration-wise and total wall-clock time improvements.

In addition, in Figure 7a we know optimizing 0.1% sensitive sparse parameters reach the performance of optimizing 10%

random subsets, and we translate such comparison to iteration-wise training time and token-wise inference time in Figure 6b.



(a) Iteration-wise & wall-clock convergence time of sensitive sparse finetuning on fixed parameters ("Sensitive") versus ZO full fine-tuning ("Full") for Llama2-7B.

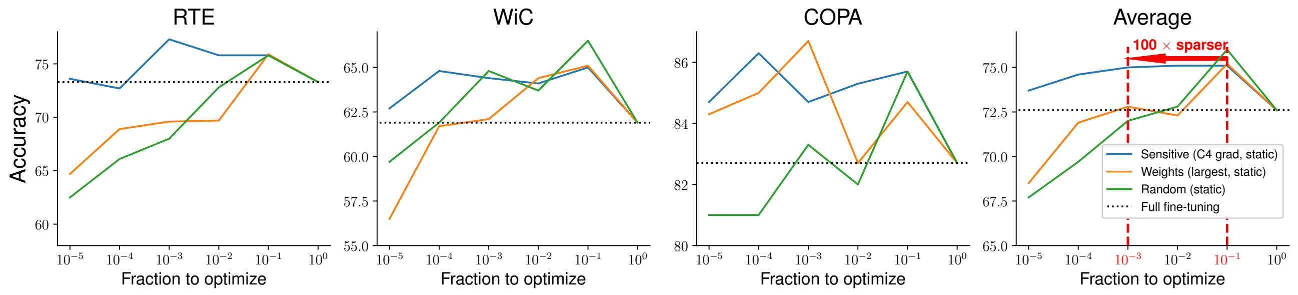


(b) Training & inference speed of Llama2-7B. As the sensitive sparse fine-tuning method achieves great performance via optimizing only *0.1%* parameters (performance comparable to ZO full-finetuning and 10% random subsets), during inference we achieve an end-to-end 1.49× speedup, with 2.15× speedup at sparse operations.

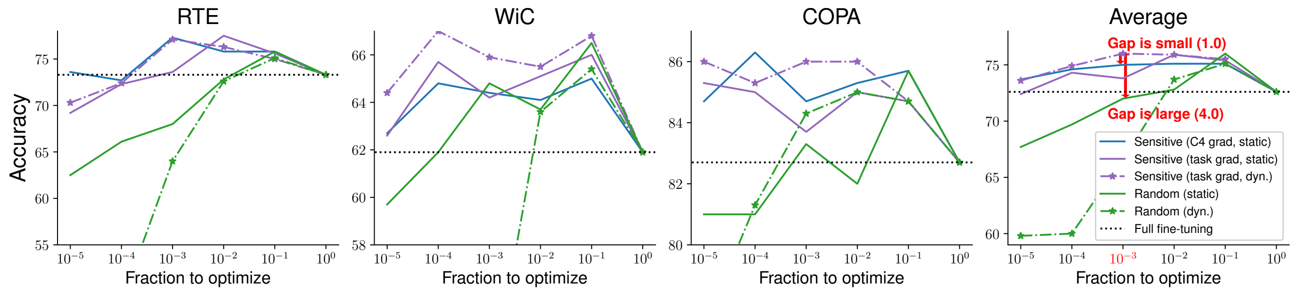
Figure 6: Wall-clock convergence time of sensitive sparse ZO optimization versus ZO full fine-tuning (subfigure 6a), and iteration-wise training and inference speedup by optimizing less parameters than random subsets (subfigure 6b).

D.4.3. Ablation study: Effectiveness of Sparse ZO Fine-Tuning on Sensitive Parameters

Here we include the complete figure of subfigure 3a and 3b in subfigure 7a and 7b. The analysis of these 2 figures is the same as our discussion in Section 4.2.



(a) Optimizing sensitive parameters with C4 gradients versus optimizing weights with largest magnitude (outliers) and random subsets of weights. The trainable parameters are all determined before fine-tuning and other parameters are kept unchanged.



(b) Optimizing sensitive parameters with C4 gradients versus task-specific gradients. "Static" means the parameters to optimize are determined before fine-tuning and other parameters are kept unchanged during fine-tuning. "Dyn." means the parameters to optimize will be updated every 100 training steps.

Figure 7: Performance of optimizing sensitive parameters in Llama2-7B fine-tuning on RTE, WiC, and COPA tasks.

D.5. Hyperparameters in experiments

For all experiments, we use 20,000 training steps with ZO-SGD optimizer (Definition 2.2). We will save a model checkpoint every 500 steps, and load the checkpoint with the lowest loss on the validation set at the end of the training, and report its test set accuracy as result. Usually, the training/validation set will be sampled from the original dataset with size 1000/500 respectively and the test set is of size $\min(1000, |\text{original test set}|)$, except for CB and COPA that we use 100 for the validation set size. For all ZO experiments (Table 5 and Table 6), we use batch size of 16. This experiment setting is identical to Malladi et al. (2023a).

Table 5: The chosen hyperparameters for experiments in Table 4. We repeat each hyperparameters for 3 random trials and report the average and standard deviation in Table 4.

(a) **Llama2-7B**

	Methods	SST-2	RTE	CB	BoolQ	WSC	WiC	COPA
Q, ZO	**Sensitive (C4, static)** (ϵ =1e-3)	5e-7	1e-6	1e-6	1e-6	5e-7	1e-6	1e-6
	LoRA (ϵ =1e-3)	1e-5	5e-5	1e-5	2e-5	1e-5	2e-5	1e-5
	Prefix (ϵ =1e-2)	1e-4	2e-4	5e-4	5e-4	1e-4	5e-4	2e-4
ZO	**Sensitive (task, static)** (ϵ =1e-3)	5e-7	1e-6	1e-6	1e-6	1e-6	1e-6	2e-6
	Random (static) (ϵ =1e-3)	2e-4	5e-4	2e-4	5e-4	2e-4	5e-4	5e-4
	Full fine-tuning (ϵ =1e-3)	5e-7	5e-7	5e-7	5e-7	2e-7	5e-7	5e-7
	ICL (#examples)	16	16	16	8	16	8	8

(b) **Mistral-7B**

	Methods	SST-2	RTE	CB	BoolQ	WSC	WiC	COPA
Q, ZO	**Sensitive (C4, static)** (ϵ =1e-4)	2e-8	5e-8	2e-8	2e-8	1e-8	2e-8	2e-8
	LoRA (ϵ =1e-4)	2e-6	5e-6	2e-6	2e-6	2e-6	2e-6	2e-6
	Prefix (ϵ =1e-3)	1e-3	2e-3	1e-3	1e-2	5e-4	1e-3	5e-4
ZO	**Sensitive (task, static)** (ϵ =1e-4)	5e-8	5e-8	2e-8	2e-8	2e-8	2e-8	2e-8
	Random (static) (ϵ =1e-4)	1e-5	2e-6	5e-6	1e-5	1e-6	2e-6	2e-5
	Full fine-tuning (ϵ =1e-4)	2e-8	2e-8	1e-8	1e-8	1e-8	1e-8	2e-8
	ICL (#examples)	4	8	4	16	4	4	8

(c) **OPT-6.7B**

	Methods	SST-2	RTE	CB	BoolQ	WSC	WiC	COPA
Q, ZO	**Sensitive (C4, static)** (ϵ =1e-3)	2e-7	5e-7	5e-7	5e-7	2e-7	5e-7	2e-7
	LoRA (ϵ =1e-3)	1e-5	2e-5	1e-5	2e-5	1e-5	2e-5	2e-5
	Prefix (ϵ =1e-2)	2e-3	1e-2	1e-3	5e-3	5e-3	1e-2	5e-3
ZO	**Sensitive (task, static)** (ϵ =1e-3)	2e-7	5e-7	5e-7	2e-7	2e-7	5e-7	2e-7
	Random (static) (ϵ =1e-3)	1e-4	5e-5	2e-5	5e-5	2e-4	5e-5	5e-5
	Full fine-tuning (ϵ =1e-3)	2e-7	2e-7	2e-7	2e-7	2e-7	2e-7	5e-7
	ICL (#examples)	16	4	16	16	16	8	16

Our hyperparameters (learning rate η, perturbation scaling constant ϵ, and the number of ICL examples) for Table 4 is reported in Table 5 for reproducibility. We use constant η and ϵ throughout our experiments. We also report the chosen hyperparameter for Figure 7a and Figure 7b in Table 6. For LoRA, we always add to all linear layers with $r = 8$ and $\alpha = 16$, and for Prefix Tuning, we always add to W_K and W_V with length as 5, as what Malladi et al. (2023a) uses.

Table 6: The chosen hyperparameters for experiments in Figure 7a and Figure 7b. We repeat each hyperparameters for 3 random trials and report the average to draw a line in Figure 7a and Figure 7b, and we use Llama2-7B for all experiments. For each subtable, we include the fraction to optimize on its header and report the chosen learning rate on each cell.

(a) **RTE**

Methods	1e-5	1e-4	1e-3	1e-2	1e-1
Sensitive (C4, static) (ϵ =1e-3)	1e-5	1e-6	1e-6	1e-6	1e-6
Sensitive (task-specific, static) (ϵ =1e-3)	1e-5	1e-6	1e-6	1e-6	1e-6
Sensitive (task-specific, dynamic) (ϵ =1e-3)	1e-5	1e-6	1e-6	1e-6	1e-6
Random (static) (ϵ =1e-3)	2e-2	5e-3	5e-4	5e-5	5e-5
Random (dynamic) (ϵ =1e-3)	2e-2	5e-3	2e-4	5e-5	5e-6
Weight outliers (static) (ϵ =1e-3)	2e-3	1e-3	2e-4	5e-5	1e-5

(b) **WiC**

Methods	1e-5	1e-4	1e-3	1e-2	1e-1
Sensitive (C4, static) (ϵ =1e-3)	1e-5	2e-6	1e-6	1e-6	1e-6
Sensitive (task-specific, static) (ϵ =1e-3)	1e-5	2e-6	1e-6	1e-6	1e-6
Sensitive (task-specific, dynamic) (ϵ =1e-3)	1e-5	2e-6	1e-6	1e-6	1e-6
Random (static) (ϵ =1e-3)	2e-2	5e-3	5e-4	5e-5	5e-6
Random (dynamic) (ϵ =1e-3)	2e-2	5e-3	5e-4	5e-5	5e-6
Weight outliers (static) (ϵ =1e-3)	1e-3	5e-4	2e-4	1e-4	2e-5

(c) **COPA**

Methods	1e-5	1e-4	1e-3	1e-2	1e-1
Sensitive (C4, static) (ϵ =1e-3)	5e-6	1e-6	1e-6	1e-6	5e-7
Sensitive (task-specific, static) (ϵ =1e-3)	5e-6	2e-6	2e-6	1e-6	1e-6
Sensitive (task-specific, dynamic) (ϵ =1e-3)	5e-6	1e-6	1e-6	1e-6	1e-6
Random (static) (ϵ =1e-3)	1e-2	2e-3	5e-4	5e-5	5e-6
Random (dynamic) (ϵ =1e-3)	2e-3	1e-3	2e-4	2e-5	2e-6
Weight outliers (static) (ϵ =1e-3)	1e-3	5e-4	5e-4	1e-4	1e-5

D.6. Task-specific prompts in experiments

We describe our task templates in Table 7.

Table 7: Task templates for all experiments. On the left column we include the task name and the model name, and on the right column we describe the exact prompt with answer candidates.

Task	Prompts
SST-2 (Llama2-7B)	### Sentence: <text> ### Sentiment: negative/positive
SST-2 (Mistral-7B, OPT-6.7B)	<text> It was terrible/great
RTE (Llama2-7B)	Suppose "<premise>" Can we infer that "<hypothesis>"? Yes or No? Yes/No
RTE (Mistral-7B, OPT-6.7B)	<premise> Does this mean that "<hypothesis>" is true? Yes or No? Yes/No
CB (Llama2-7B, Mistral-7B, OPT-6.7B)	Suppose <premise> Can we infer that "<hypothesis>"? Yes, No, or Maybe? Yes/No/Maybe
BoolQ (Llama2-7B)	<passage> <question>? Yes/No
BoolQ (Mistral-7B, OPT-6.7B)	<passage> <question>? Yes/No
WSC (Llama2-7B, Mistral-7B, OPT-6.7B)	<text> In the previous sentence, does the pronoun "<span2>" refer to <span1>? Yes or No? Yes/No
WiC (Llama2-7B, Mistral-7B, OPT-6.7B)	Does the word "<word>" have the same meaning in these two sentences? Yes, No? <sent1> <sent2> Yes/No
COPA (Llama2-7B, Mistral-7B, OPT-6.7B)	<premise> so/because <candidate>

D.7. Implementation of sparse operations in linear layers

Linear layers in LLMs often contribute most parameters (Kaplan et al., 2020). Since from Equation 5 we know

$$\mathbf{w}_{\text{sparse}} = \mathbf{w} \odot \mathbf{m}_k, \quad \mathbf{w}_{\text{dense}} = \mathbf{w} \odot (\mathbf{1}_d - \mathbf{m}_k), \quad \mathbf{w} = \mathbf{w}_{\text{sparse}} + \mathbf{w}_{\text{dense}} \tag{16}$$

and since $\mathbf{w}_{\text{dense}}$ would have same shape (and same computational intensities) as $\mathbf{w}$, we need to improve *wall-clock time* efficiency of $\mathbf{w}_{\text{sparse}}\mathbf{x}$ to improve the computational efficiency of linear layers after extracting the sparse parameters. In this case, we would have two different methods to implement the forward pass of linear layers (with induced sparse operation colored in red):

$$\begin{aligned}
\mathbf{w}\mathbf{x} &= \mathbf{w}_{\text{dense}}\mathbf{x} + \mathbf{w}_{\text{sparse}}\mathbf{x} && && (17)\\
&= \text{SparseAddMM}(\text{DenseMM}(\mathbf{w}_{\text{dense}}, \mathbf{x}), \mathbf{w}_{\text{sparse}}, \mathbf{x}) && \text{faster with token generation} && (18)\\
&= (\mathbf{w}_{\text{dense}} + \mathbf{w}_{\text{sparse}})\mathbf{x} && && (19)\\
&= \text{DenseMM}(\text{SparseAdd}(\mathbf{w}_{\text{sparse}}, \mathbf{w}_{\text{dense}}), \mathbf{x}) && \text{faster with ZO training} && (20)
\end{aligned}$$

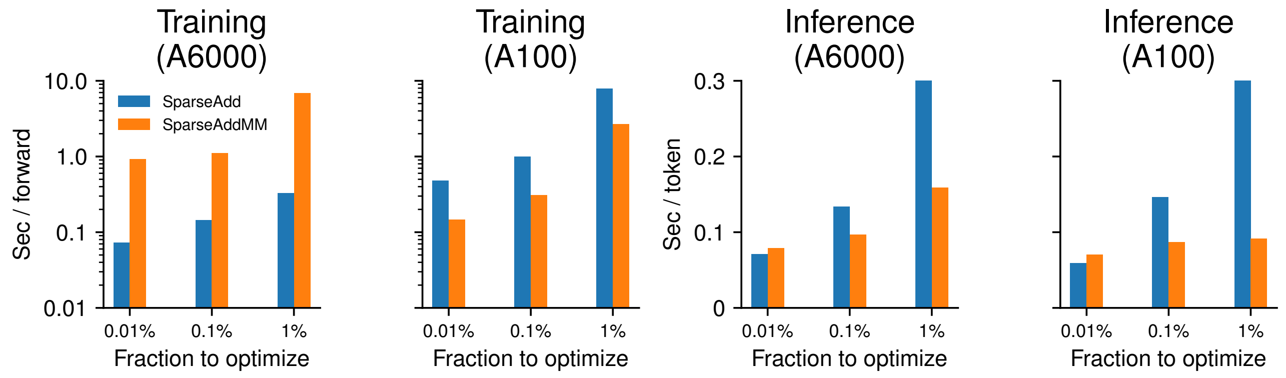


Figure 8: Time of SparseAdd (Equation 20) versus SparseAddMM (Equation 18) in Llama2-7B ZO training forward & inference. In subfigure 1 and 3, we use Nvidia RTX A6000 and Intel Xeno Gold 6342 CPUs, with PyTorch version 2.2, HuggingFace version 4.36, and CUDA 12.2. In subfigure 2 and 4, we use Nvidia A100-SXM4 (40 GiB) and AMD EPYC 7543P 32-Core CPU with PyTorch version 2.1, HuggingFace version 4.38.2, and CUDA 12.2. We use Flash Attention 2 (Dao, 2023) for all 4 subfigures.

The specific choice of employing Equation 18 or Equation 20 needs careful consideration and benchmarking, but here we can provide a general guideline based on the size of input vector (or arithmetic intensity) and potential integration with weight quantization method:

Size of input vectors $\mathbf{x}$ and arithmetic intensity. $\mathbf{w}_{\text{sparse}}\mathbf{x}$ in Equation 18 would have a computational dependency over $\mathbf{x}$. During large-batched ZO training, $\mathbf{x}$ would be large enough such that Equation 18 would induce large computational overhead, as shown in subfigure 1 of Figure 8. In contrast, the computational complexity of Equation 20 is *independent* of $\mathbf{x}$ and when $\mathbf{x}$ is large, we would expect Equation 20 is *much* faster than Equation 18. As an example, we use sequence length of 512 and batch size 16 sampled from WikiText-2 dataset (Merity et al., 2016) as a representative computational intensity for ZO training in subfigures 1 and 2 in Figure 8.

However, during autoregressive token generation, on each step we would only append *a single token* to the previously cached embeddings, and in this case $\mathbf{x}$ is small and computing $\mathbf{w}_{\text{dense}} + \mathbf{w}_{\text{sparse}}$ is generally not worthwhile, especially given that $\mathbf{w}_{\text{sparse}}$ is already sparse. This is also illustrated in subfigure 3 and 4 in Figure 8. However, we note that the specific implementation choice is hardware and task dependent and requires thorough benchmarking and we will leave it as a future work.

We recommend using Equation 20 during large-batched ZO training and Equation 18 during small-batched autoregressive token generation.

In light of this observation, in our Figure 6b, we implement both “SparseAdd” and “SparseAddMM” methods for “Sensitive (0.1%)” and “Random (10%)”. For each method we report the *lowest* time out of these 2 implementations: for “Sensitive (0.1%)” training and “Random (10%)” training and inference, we use “SparseAdd” approach. For “Sensitive (0.1%)” inference, we use the “SparseAddMM” approach.

Integration with weight quantization method. Weight quantization algorithms can be categorized into 2 categories: uniform quantization method and non-uniform quantization method. For uniform quantization method, (Xi et al., 2023) indicates that we could use integer matrix multiplication to compute $Q(\mathbf{w}_{\text{dense}})\mathbf{x}$ efficiently *without* first dequantizing $Q(\mathbf{w}_{\text{dense}})$ to 16 bits. However, this creates difficulty on our “SparseAdd” approach as we will *violate the constraint of uniformly-spaced quantization bins* by computing $\text{SparseAdd}(Q(\mathbf{w}_{\text{dense}}) + \mathbf{w}_{\text{sparse}})$. In this case, we also have 3 different implementations:

$$\begin{aligned}
Q(\mathbf{w})\mathbf{x} &\sim Q(\mathbf{w}_{\text{dense}})\mathbf{x}+\mathbf{w}_{\text{sparse}}\mathbf{x} && (21)\\
&= \text{SparseAddMM}\Big(\text{Dequantize}\big(\text{IntMM}(Q(\mathbf{w}_{\text{dense}}), \mathbf{x})\big), \mathbf{w}_{\text{sparse}}, \mathbf{x}\Big) \quad \text{fits with integer matmul} && (22)\\
&= \text{SparseAddMM}\Big(\text{Dequantize}(Q(\mathbf{w}_{\text{dense}})), \mathbf{x}, \mathbf{w}_{\text{sparse}}\Big) \quad \text{similar to Equation 18} && (23)\\
&= (\text{Dequantize}(Q(\mathbf{w}_{\text{dense}}))+\mathbf{w}_{\text{sparse}})\mathbf{x} && (24)\\
&= \text{DenseMM}(\text{SparseAdd}\ (\mathbf{w}_{\text{sparse}}, \text{Dequantize}(Q(\mathbf{w}_{\text{dense}})), \mathbf{x}) \quad \text{similar to Equation 20} && (25)
\end{aligned}$$

Equation 22 would compute $\text{IntMM}(Q(\mathbf{w}_{\text{dense}}), \mathbf{x})$ *before* dequantizing it to 16 bits. This would make “SparseAdd” approach infeasible and we can only employ “SparseAddMM” approach in this case. Notice that both Equation 23 and Equation 25 would still dequantize $Q(\mathbf{w}_{\text{dense}})$ first and the choice of implementation would follow into our discussion of input vector size $\mathbf{x}$ in last paragraph. We leave a practical implementation and thorough benchmarking into a future work.

We recommend using Equation 22 when we use efficient integer matmul to compute $Q(\mathbf{w}_{\text{dense}})\mathbf{x}$ and in other cases, using Equation 23 or Equation 25 follows our previous recommendation based on the size of input vectors.

D.8. Hardware, platform, libraries, and other details for fine-tuning and benchmarking

Figure 6a, Figure 6b, and Figure 8 (subfigure 1 and 3) are trained and evaluated on an internal cluster with 8 Nvidia RTX A6000 GPUs and 2 Intel Xeon Gold 6342 CPUs, with PyTorch version 2.2, HuggingFace version 4.36, and CUDA 12.2. In subfigure 2 and 4 in Figure 8, we use Nvidia A100-SXM4 (40 GiB) and AMD EPYC 7543P 32-Core CPU with PyTorch version 2.1, HuggingFace version 4.38.2, and CUDA 12.2. We use Flash Attention 2 (Dao, 2023) in HuggingFace Transformers library throughout our experiments, and the base model for ZO full fine-tuning and benchmarking is always Llama2-7B with Float16 datatype (torch.float16). We also use the Float16 datatype (torch.float16) for all of our sparse parameters (sensitive sparse, random subsets, etc.) in ZO fine-tuning experiments. Notice that for all of the FO fine-tuning demonstrations (Figure 4 and Figure 5) we use the BrainFloat16 datatype (torch.bfloat16) to avoid the NaN issue from the Float16 datatype.

In Figure 6b and Figure 8, we use sequence length of 512 and batch size 16 sampled from WikiText-2 dataset (Merity et al., 2016) as a representative computational intensity for ZO training, and for inference we generate 128 tokens with top-p ($p = 0.9$) sampling from the prompt “*Please describe the effect of sparse zeroth-order optimization methods on memory-efficient LLM fine-tuning:* ”.